Daniel Carpenter

OU ID: 113009743 | daniel.carpenter@ou.edu

Individual Project

**Job-Shop Accounting Database Implementation**

Database Management Systems (DSA 4513-995)

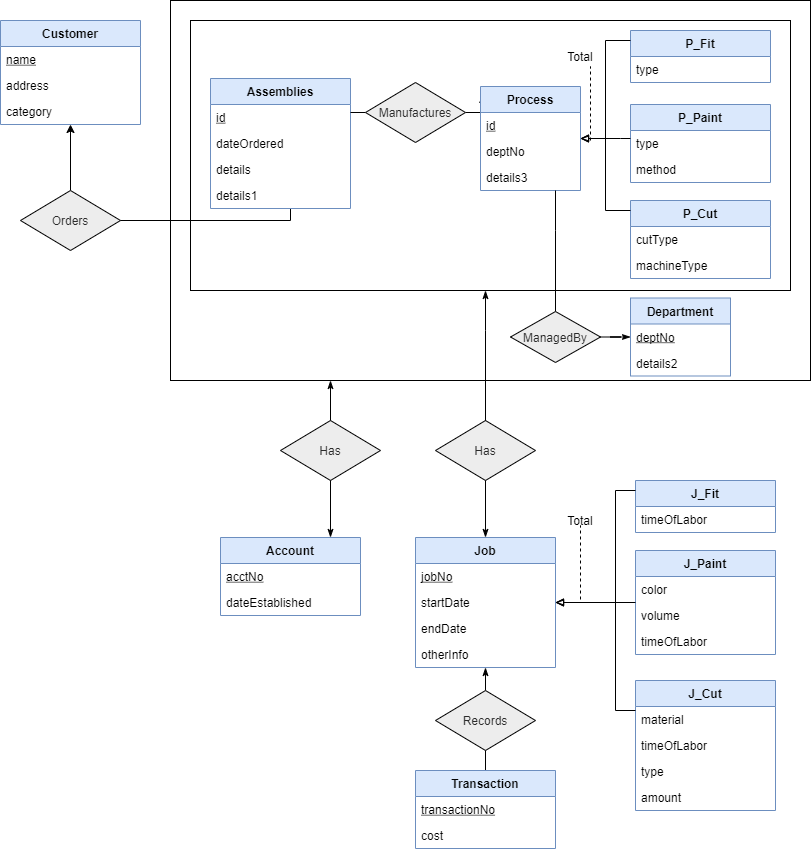
Fall 2021

Instructor Dr. Le Gruenwald

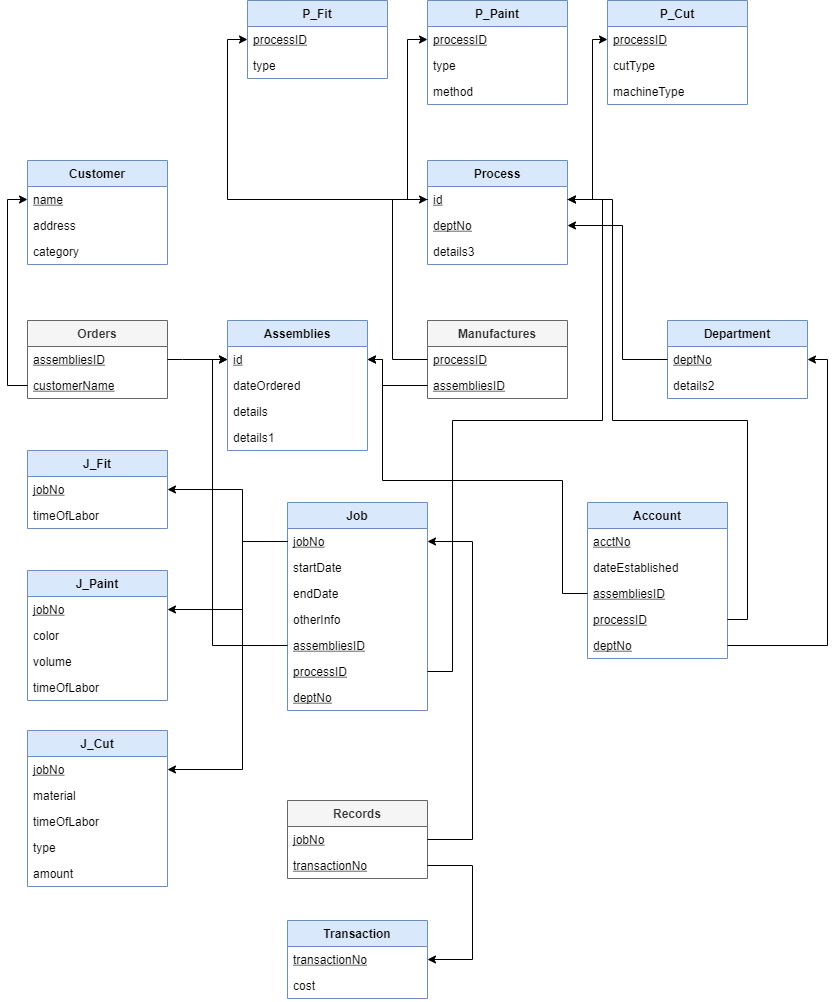
**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Tasks Performed** | | **Page Number** |
| *Task 1.* |  | *2-3* |
|  | 1.1. Entity Relation Diagram (ERD) | 2-2 |
|  | 1.2. Relational Database Schema | 3-3 |
| *Task 2.* | *Data Dictionary* | *4-5* |
| *Task 3.* |  | *6-10* |
|  | 3.1. Storage Structures for Tables | 6-9 |
|  | 3.2. Storage Structures for Tables in Azure SQL Database | 10-10 |
| *Task 4.* | *SQL Statements for Creation of Tables in Azure SQL Database* | *11-17* |
| *Task 5.* |  | *18-51* |
|  | 5.1. SQL Statements and Associated Stored Procedures Implementing All Queries (1-15 and Error Checking) | 18-30 |
|  | 5.2. Java Source Program and Associated Screenshots of Successful Compilation | 31-51 |
| *Task 6.* | *Java Program Execution* | *52-79* |
|  | 6.1. Screenshots showing the testing of Query 1 | 52-52 |
|  | 6.2. Screenshots showing the testing of Query 2 | 53-53 |
|  | 6.3. Screenshots showing the testing of Query 3 | 54-55 |
|  | 6.4. Screenshots showing the testing of Query 4 | 56-56 |
|  | 6.5. Screenshots showing the testing of Query 5 | 57-57 |
|  | 6.6. Screenshots showing the testing of Query 6 | 58-59 |
|  | 6.7. Screenshots showing the testing of Query 7 | 60-60 |
|  | 6.8. Screenshots showing the testing of Query 8 | 61-61 |
|  | 6.9. Screenshots showing the testing of Query 9 | 62-62 |
|  | 6.10. Screenshots showing the testing of Query 10 | 63-63 |
|  | 6.11. Screenshots showing the testing of Query 11 | 64-64 |
|  | 6.12. Screenshots showing the testing of Query 12 | 65-65 |
|  | 6.13. Screenshots showing the testing of Query 13 | 66-66 |
|  | 6.14. Screenshots showing the testing of Query 14 | 67-69 |
|  | 6.15. Screenshots showing the testing of Query 15 | 70-72 |
|  | 6.16. Screenshots showing the testing of the import and export options | 73-75 |
|  | 6.17. Screenshots showing the testing of three types of errors | 76-78 |
|  | 6.18. Screenshots showing the testing of the quit option | 79-79 |
| *Task 7.* | *Web database application and its execution* | *80-95* |
|  | 7.1. Web Database Application Source Program and Associated Screenshots of Successful Compilation | 80-90 |
|  | 7.2. Screenshots Showing the Testing of the Web Database Application | 91-96 |

**1.1. Entity Relation Diagram (ERD)**



**1.2. Relational Database Schema**



**Task 2. Data Dictionary**



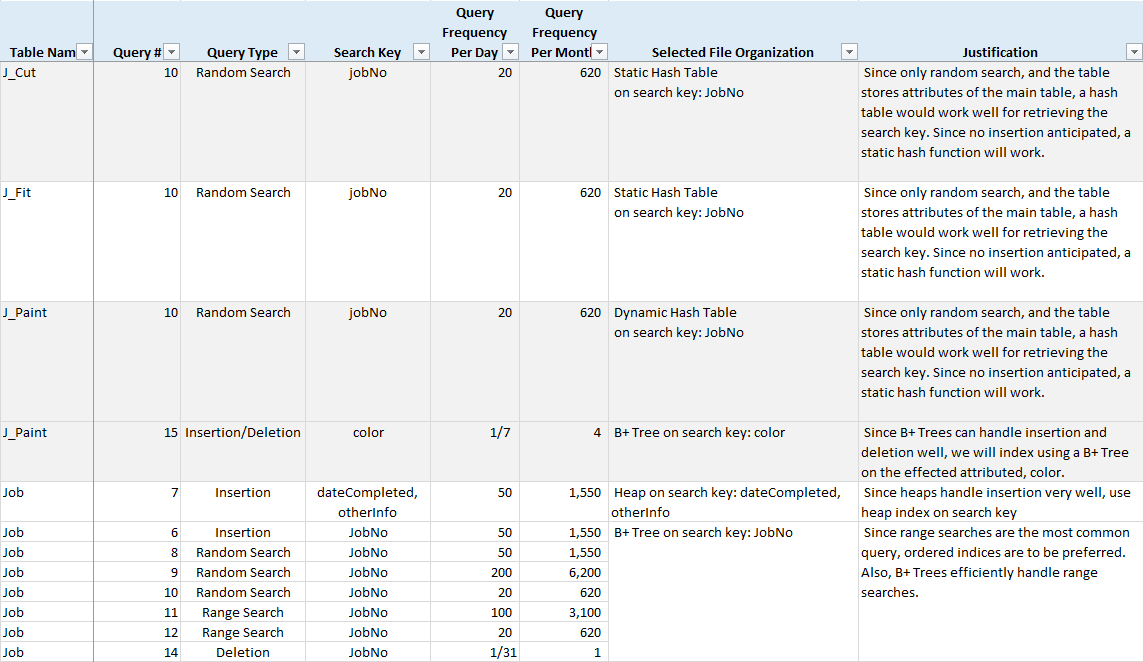
**Task 2. Data Dictionary *(Continued)***



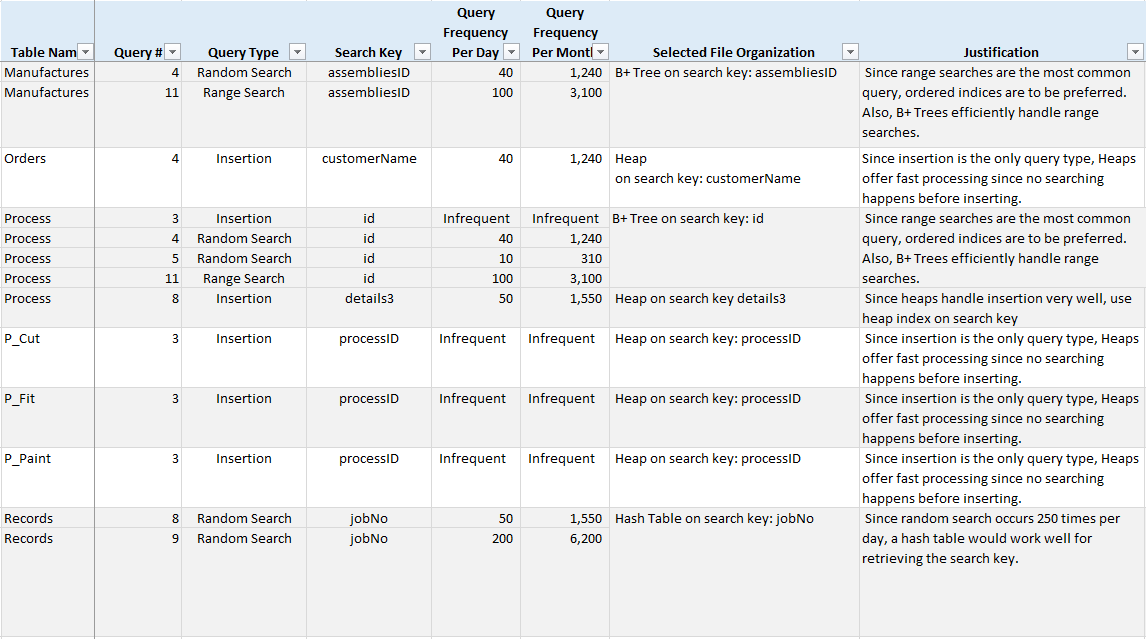
**3.1. Storage Structures for Tables**



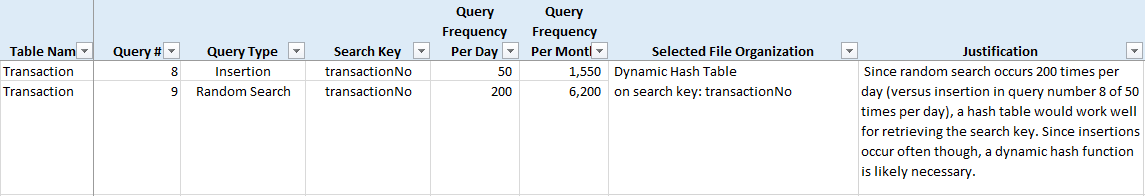
**3.1. Storage Structures for Tables *(Continued)***

****

**3.1. Storage Structures for Tables *(Continued)***

****

**3.1. Storage Structures for Tables *(Continued)***

****

**3.2. Storage Structures for Table (Azure SQL Database)**

**B+ Tree Data Structure**

In order to implement a B+ Tree, basic index creation in SQL Server are B+ Trees. Therefore, for all recommended indices with B+ Tree implementations use a simple index on the attribute.

Source: <https://sqlity.net/en/2445/b-plus-tree/>

**Heap Data Structure**

By default, a table is a Heap when created.

Source:<https://docs.microsoft.com/en-us/sql/relational-databases/indexes/heaps-tables-without-clustered-indexes?view=sql-server-ver15>

**Static Hash Table Data Structure**

It is possible to create Hash tables for memory optimized tables. To implement this hash index, you would alter (either immediately after creation of the table, or at some future point) the table and specify the number of buckets for each bin. However, I am unaware of how to create this memory optimized table; per recommendations of Microsoft for an alternative, I have instead created a non-clustered index on the specified attribute(s).

Source: <https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15#hash_index>

**Dynamic Hash Table Data Structure**

The process is the same as creating a static hash index, but you would dynamically set the number of buckets (or bins). Note that you would need to somehow schedule some update of these bins, which I theorize could be possible with a scheduled stored procedure. If I were to implement it, I would use some hash function based on the number of bins (e.g. using a modulo function) for the table at that point in time. Due to the issues stated in the Static Hash section, I have chosen to use a non-clustered index for each specified attribute.

**Task 4. SQL Statements to Create Tables in Azure SQL Database**

|  |
| --- |
| *-- ==========================================================================*  *-- @class:  DSA 4513*  *-- @asnmt:  Class Project*  *-- @task:   4*  *-- @author: Daniel Carpenter, ID: 113009743*  *-- @description:*  *--     Queries to create tables, contraints, and indiceds of the job-shop*  *--     accounting system database*  *-- ==========================================================================*  *-- Use Daniel Carpenter's Azure Database for all queries*  USE [cs-dsa-4513-sql-db]  *-- ==========================================================================*  *-- DROP TABLES IF ALREADY EXISTING*  *-- Note: Using schema named 'Project' for this Project's Tables*  *-- ==========================================================================*  *-- Tables ---------------------------------------------------------------*      DROP TABLE IF EXISTS [Project].[P\_Fit];      DROP TABLE IF EXISTS [Project].[P\_Paint];      DROP TABLE IF EXISTS [Project].[P\_Cut];      DROP TABLE IF EXISTS [Project].[J\_Fit];      DROP TABLE IF EXISTS [Project].[J\_Paint];      DROP TABLE IF EXISTS [Project].[J\_Cut];      DROP TABLE IF EXISTS [Project].[Orders];      DROP TABLE IF EXISTS [Project].[Manufactures];      DROP TABLE IF EXISTS [Project].[ManagedBy];      DROP TABLE IF EXISTS [Project].[Records];      DROP TABLE IF EXISTS [Project].[Customer];      DROP TABLE IF EXISTS [Project].[Job];      DROP TABLE IF EXISTS [Project].[Transaction];      DROP TABLE IF EXISTS [Project].[Account];      DROP TABLE IF EXISTS [Project].[Assemblies];      DROP TABLE IF EXISTS [Project].[Process];      DROP TABLE IF EXISTS [Project].[Department];  *-- ==========================================================================*  *-- CREATE TABLES, INDICES, AND CONSTRAINTS FOR JOB-SHOP DATABASE*  *-- ==========================================================================*  *-- Create the Customer Table ------------------------------------------------*  CREATE TABLE [Project].Customer (      [name]          VARCHAR(255) PRIMARY KEY,      [address]       VARCHAR(255) NOT NULL,      [category]      INT NOT NULL,      CONSTRAINT      CTGRY\_RANGE CHECK(category >= 1 AND category <= 10)  )  *-- Create B+ Tree Index on name of Customer table*  CREATE INDEX [IDX\_Customer\_ON\_name] ON [Project].Customer([name]);  *-- Create the Assemblies Table ----------------------------------------------*  CREATE TABLE [Project].Assemblies (      [id]            INT PRIMARY KEY,      [dateOrdered]   DATE NOT NULL,      [details]       VARCHAR(255) DEFAULT NULL,      [details1]      REAL DEFAULT 0,      CONSTRAINT      [NON\_NEG\_id\_Assemblies] CHECK(id > 0)  )  *-- Create B+ Tree Index on id of Assemblies table*  CREATE INDEX [IDX\_Assemblies\_ON\_id] ON [Project].Assemblies(id);  *-- Create the Orders Table --------------------------------------------------*  CREATE TABLE [Project].Orders (      [customerName]  VARCHAR(255) FOREIGN KEY REFERENCES [Project].Customer,      [assembliesID]  INT NOT NULL FOREIGN KEY REFERENCES [Project].Assemblies  )  *-- Create the Fit Table for Processes ---------------------------------------*  CREATE TABLE [Project].P\_Fit (      [processID]             INT PRIMARY KEY,      [type]          VARCHAR(255) DEFAULT NULL,      CONSTRAINT      [NON\_NEG\_id\_P\_Fit] CHECK(processID > 0)  )  *-- Create the Paint Table for Processes -------------------------------------*  CREATE TABLE [Project].P\_Paint (      [processID]             INT PRIMARY KEY,      [type]          VARCHAR(255) DEFAULT NULL,      [method]        VARCHAR(255) DEFAULT NULL,      CONSTRAINT      [NON\_NEG\_id\_P\_Paint] CHECK(processID > 0)  )  *-- Create the Cut Table for Processes ---------------------------------------*  CREATE TABLE [Project].P\_Cut (      [processID]             INT PRIMARY KEY,      [cutType]       VARCHAR(255) DEFAULT NULL,      [machineType]   VARCHAR(255) DEFAULT NULL,      CONSTRAINT      [NON\_NEG\_id\_P\_Cut] CHECK(processID > 0)  )  *-- Create the Table for Departments------------------------------------------*  CREATE TABLE [Project].Department (      [deptNo]        INT PRIMARY KEY,      [details2]      REAL DEFAULT 0,      CONSTRAINT      [NON\_NEG\_deptNo\_Department] CHECK(deptNo > 0)  )  *-- Create B+ Tree Index on deptNo of the Department table*  CREATE INDEX [IDX\_Department\_ON\_deptNo] ON [Project].Department(deptNo);  *-- Create the Table for Processes -------------------------------------------*  CREATE TABLE [Project].Process (      [id]            INT PRIMARY KEY,      [deptNo]        INT FOREIGN KEY REFERENCES [Project].Department,      [details3]      REAL DEFAULT 0,      CONSTRAINT      [NON\_NEG\_id\_Process] CHECK(id > 0)  )  *-- Create the Relation Table between the Process and Assemblies tables ------*  CREATE TABLE [Project].Manufactures (      [assembliesID]  INT NOT NULL FOREIGN KEY REFERENCES [Project].Assemblies,      [processID]     INT NOT NULL FOREIGN KEY REFERENCES [Project].Process  )  *--Create B+ Tree Index on deptNo of the Manufactures table*  CREATE INDEX [IDX\_Manufactures\_ON\_assembliesID] ON [Project].Manufactures(assembliesID);  *-- Create the Table for Jobs ------------------------------------------------*  CREATE TABLE [Project].Job (      [jobNo]         INT PRIMARY KEY,      [startDate]     DATE NOT NULL,      [endDate]       DATE DEFAULT NULL,      [otherInfo]     VARCHAR(255) DEFAULT NULL,      [assembliesID]  INT NOT NULL FOREIGN KEY REFERENCES [Project].Assemblies,      [processID]     INT NOT NULL FOREIGN KEY REFERENCES [Project].Process,      [deptNo]        INT NOT NULL FOREIGN KEY REFERENCES [Project].Department,      CONSTRAINT      [NON\_NEG\_JobNo\_Job] CHECK(JobNo > 0),      CONSTRAINT      [POS\_VAR\_DATES\_Job] CHECK(endDate >= startDate)  )  *-- Create B+ Tree Index on id of Job table*  CREATE INDEX [IDX\_Job\_ON\_JobNo] ON [Project].Job(JobNo);  *-- Create the Fit Table for Jobs --------------------------------------------*  CREATE TABLE [Project].J\_Fit (      [jobNo]         INT FOREIGN KEY REFERENCES [Project].Job,      [timeOfLabor]   INT DEFAULT 0,      CONSTRAINT      [NON\_NEG\_id\_J\_Fit] CHECK(jobNo > 0),      CONSTRAINT      [NON\_NEG\_timeOfLabor\_J\_Fit] CHECK([timeOfLabor] >= 0)  )  *-- Create non-clustered index in place of ideal static hash table*  CREATE NONCLUSTERED INDEX [IDX\_J\_Fit\_ON\_id] ON [Project].J\_Fit(jobNo);  *-- Create the Paint Table for Jobs ------------------------------------------*  CREATE TABLE [Project].J\_Paint (      [jobNo]         INT FOREIGN KEY REFERENCES [Project].Job,      [color]         VARCHAR(255) DEFAULT NULL,      [volume]        REAL DEFAULT 0 NOT NULL,      [timeOfLabor]   INT DEFAULT 0 NOT NULL,      CONSTRAINT      [NON\_NEG\_id\_J\_Paint] CHECK(jobNo > 0),      CONSTRAINT      [NON\_NEG\_timeOfLabor\_J\_Paint] CHECK([timeOfLabor] >= 0)  )  *-- Create non-clustered index in place of dunamic hash table on id*  CREATE NONCLUSTERED INDEX [IDX\_J\_Paint\_ON\_id] ON [Project].J\_Paint(jobNo);  *-- Create B+ tree index for color*  CREATE INDEX [IDX\_J\_Paint\_ON\_color] ON [Project].J\_Paint(color);  *-- Create the Cut Table for Jobs --------------------------------------------*  CREATE TABLE [Project].J\_Cut (      [jobNo]         INT FOREIGN KEY REFERENCES [Project].Job,      [material]      VARCHAR(255) DEFAULT NULL,      [timeOfLabor]   INT DEFAULT 0 NOT NULL,      [type]          VARCHAR(255) DEFAULT NULL,      [amount]        REAL DEFAULT 0 NOT NULL,      CONSTRAINT      [NON\_NEG\_id\_J\_Cut]          CHECK(jobNo > 0),      CONSTRAINT      [NON\_NEG\_timeOfLabor\_J\_Cut] CHECK([timeOfLabor] >= 0)  )  *-- Create non-clustered index in place of ideal static hash table*  CREATE NONCLUSTERED INDEX [IDX\_J\_Cut\_ON\_id] ON [Project].J\_Cut(jobNo);  *-- Create the Table for Transactions -----------------------------------------*  CREATE TABLE [Project].[Transaction] (      [transactionNo] INT PRIMARY KEY,      [cost]          REAL DEFAULT 0 NOT NULL,      CONSTRAINT      [NON\_NEG\_transactionNo\_Transaction] CHECK(transactionNo > 0),      CONSTRAINT      [NON\_NEG\_cost\_Transaction]          CHECK(cost >= 0)  )  *-- Create non-clustered index in place of ideal dynamic hash table*  CREATE NONCLUSTERED INDEX [IDX\_Transaction\_ON\_transactionNo] ON [Project].[Transaction](transactionNo);  *-- Create the Relation Table for the Job and Transactions tables -------------*  CREATE TABLE [Project].Records (      [jobNo]         INT NOT NULL FOREIGN KEY REFERENCES [Project].Job,      [transactionNo] INT NOT NULL FOREIGN KEY REFERENCES [Project].[Transaction]  )  *-- Create non-clustered index in place of ideal dynamic hash table*  CREATE NONCLUSTERED INDEX [IDX\_Records\_ON\_jobNo] ON [Project].Records(jobNo);  *-- Create the Table for Accounts ---------------------------------------------*  CREATE TABLE [Project].Account (      [acctNo]            INT PRIMARY KEY,      [dateEstablished]   DATE NOT NULL,      [assembliesID]      INT NOT NULL FOREIGN KEY REFERENCES [Project].Assemblies,      [processID]         INT NOT NULL FOREIGN KEY REFERENCES [Project].Process,      [deptNo]            INT NOT NULL FOREIGN KEY REFERENCES [Project].Department,      CONSTRAINT          [NON\_NEG\_acctNo\_Account] CHECK(acctNo > 0)  )  *-- Create B+ Tree Index on id of Accounts table*  CREATE INDEX [IDX\_Account\_ON\_acctNo] ON [Project].Account(acctNo); |

**Task 4. Continued: SQL Table Creation Successful Compilation Screenshot**

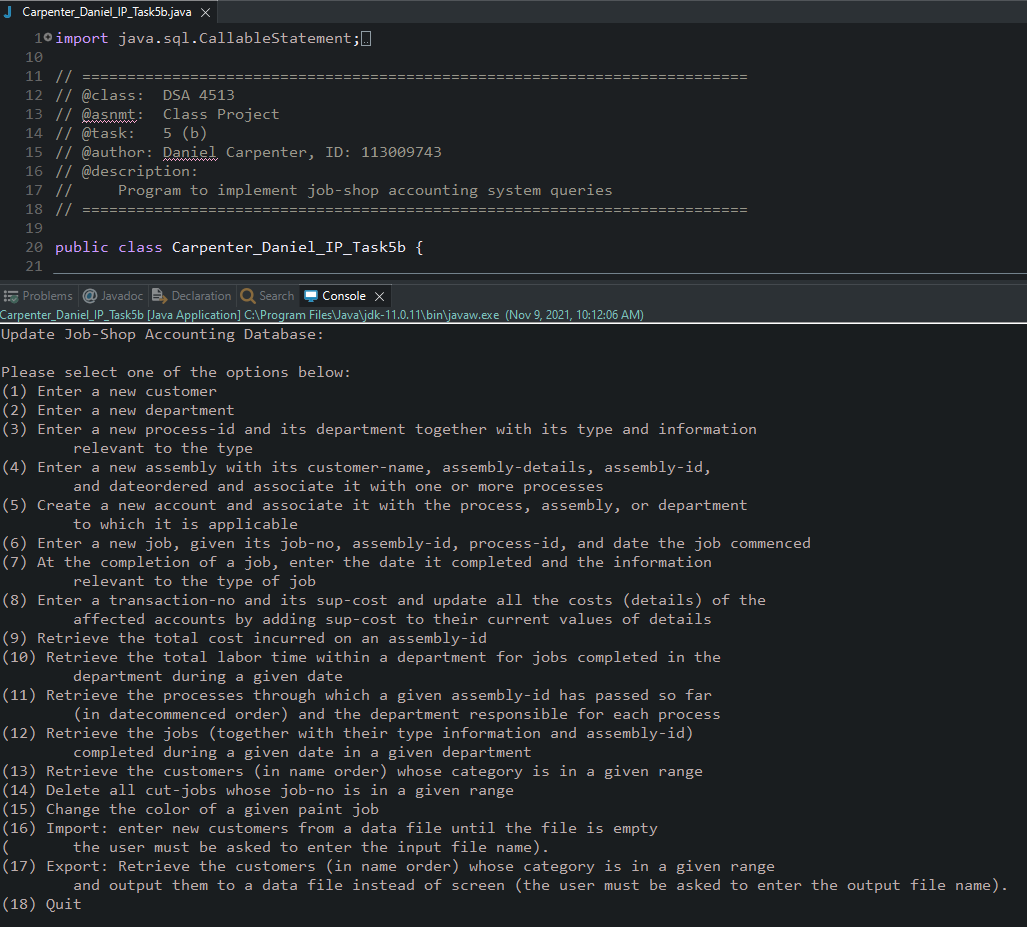
**5.1. SQL Statements and Stored Procedures Implementing Queries 1 – 15**

|  |
| --- |
| *-- ==========================================================================*  *-- @class:  DSA 4513*  *-- @asnmt:  Class Project*  *-- @task:   5 (a)*  *-- @author: Daniel Carpenter, ID: 113009743*  *-- @description:*  *--     Queries to create procedures for queries 1 - 15 and 17 of the*  *--     job-shop accounting system database*  *-- ==========================================================================*  *-- Use Daniel Carpenter's Azure Database for all queries*  USE [cs-dsa-4513-sql-db]  *-- ==========================================================================*  *-- DROP PROCEDURES IF ALREADY EXISTING*  *-- Note: Using schema named 'Project' for this Project's Tables*  *-- ==========================================================================*  *-- Drop Procedures  -----------------------------------------------------*      DROP PROCEDURE IF EXISTS [Project].addCustomer;         *-- 1*      DROP PROCEDURE IF EXISTS [Project].addDepartment;       *-- 2*      DROP PROCEDURE IF EXISTS [Project].addProcess;          *-- 3*      DROP PROCEDURE IF EXISTS [Project].addAssembly;         *-- 4*      DROP PROCEDURE IF EXISTS [Project].addAccount;          *-- 5*      DROP PROCEDURE IF EXISTS [Project].addJob;              *-- 6*      DROP PROCEDURE IF EXISTS [Project].setJobAsCompleted;   *-- 7*      DROP PROCEDURE IF EXISTS [Project].addTransaction;      *-- 8*      DROP PROCEDURE IF EXISTS [Project].getTotalCosts;       *-- 9*      DROP PROCEDURE IF EXISTS [Project].getTotalLaborTime;   *-- 10*      DROP PROCEDURE IF EXISTS [Project].getProcessUpdate;    *-- 11*      DROP PROCEDURE IF EXISTS [Project].getJobs;             *-- 12*      DROP PROCEDURE IF EXISTS [Project].getCustomers;        *-- 13*      DROP PROCEDURE IF EXISTS [Project].deleteJobs;          *-- 14*      DROP PROCEDURE IF EXISTS [Project].setPaintJob;         *-- 15*      DROP PROCEDURE IF EXISTS [Project].getCustomersInRange; *-- 17*  *-- ==========================================================================*  *-- CREATE PROCDURES USED IN JAVA*  *-- ==========================================================================*  *-----------------------------------------------------------------------------*  *-- 1. Enter a new customer*  *-----------------------------------------------------------------------------*        GO      CREATE PROCEDURE [Project].addCustomer          @name       VARCHAR(255),          @address    VARCHAR(255),          @category   INT          AS          BEGIN              INSERT INTO [Project].Customer VALUES (@name, @address, @category)          END      GO  *-----------------------------------------------------------------------------*  *-- 2. Enter a new department*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].addDepartment          @deptNo INT          AS          BEGIN              INSERT INTO [Project].Department VALUES (@deptNo, 0)          END      GO  *-----------------------------------------------------------------------------*  *-- 3. Enter a new process-id and its department together with its type and information*  *-- relevant to the type*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].addProcess          @id         INT,          @deptNo     INT          AS          BEGIN  *-- Update the process table*              INSERT INTO [Project].Process VALUES (@id, @deptNo, 0) *-- assumes that the department already exists*  *-- Insert the Process into Job Cut, Fit, and Paint table*  *-- with other attributes assumed null or 0 where applicable*              INSERT INTO [Project].[P\_Cut]   (processID) VALUES (@id)              INSERT INTO [Project].[P\_Fit]   (processID) VALUES (@id)              INSERT INTO [Project].[P\_Paint] (processID) VALUES (@id)          END      GO  *-----------------------------------------------------------------------------*  *-- 4. Enter a new assembly with its customer-name, assembly-details, assembly-id,*  *-- and dateordered and associate it with one or more processes*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].addAssembly          @assID          INT,          @dateOrdered    DATE,          @details        VARCHAR(255),          @customerName   VARCHAR(255),          @processID      INT          AS          BEGIN  *-- Add the assembly*              INSERT INTO [Project].Assemblies VALUES (@assID, @dateOrdered, @details, 0)    *-- Add assembly id and customer name to the relation table 'Orders'*              INSERT INTO [Project].Orders     VALUES (@customerName, @assID)  *-- Update into the Relation table manufactures*              INSERT INTO [Project].Manufactures VALUES (@assID, @processID)          END      GO  *-----------------------------------------------------------------------------*  *-- 5. Create a new account and associate it with the process, assembly, or department*  *-- to which it is applicable*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].addAccount          @acctNo             INT,          @dateEstablished    DATE,          @assembliesID       INT,          @processID          INT,          @deptNo             INT          AS          BEGIN  *-- Add an account to the 'Account' table*              INSERT INTO [Project].Account VALUES (@acctNo, @dateEstablished, @assembliesID, @processID, @deptNo)            END      GO    *-----------------------------------------------------------------------------*  *-- 6. Enter a new job, given its job-no, assembly-id, process-id, and date the job commenced*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].addJob          @jobNo          INT,          @startDate      DATE,          @assembliesID   INT,          @processID      INT,          @deptNo        INT          AS          BEGIN  *-- Insert only releveant values into Job table,*  *-- intentionally omitting end date and other info attributes*              INSERT INTO [Project].Job (jobNo,   startDate,  assembliesID,  processID,  deptNo)              VALUES                    (@jobNo, @startDate, @assembliesID, @processID, @deptNo)  *-- Insert the job into Job Cut, Fit, and Paint table*  *-- with other attributes assumed null or 0 where applicable*              INSERT INTO [Project].[J\_Cut]   (jobNo) VALUES (@jobNo)              INSERT INTO [Project].[J\_Fit]   (jobNo) VALUES (@jobNo)              INSERT INTO [Project].[J\_Paint] (jobNo) VALUES (@jobNo)          END      GO  *-----------------------------------------------------------------------------*  *-- 7. At the completion of a job, enter the date it completed and the information*  *-- relevant to the type of job*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].setJobAsCompleted          @jobNo      INT,          @endDate    DATE,          @otherInfo  VARCHAR(255)            AS          BEGIN  *-- Update existing job's end date and other info*              UPDATE  [Project].Job              SET [Project].Job.endDate   = @endDate,                  [Project].Job.otherInfo = @otherInfo  *-- only show selected values for a given job number*              WHERE [Project].Job.jobNo   = @jobNo          END      GO  *-----------------------------------------------------------------------------*  *-- 8. Enter a transaction-no and its sup-cost and update all the costs (details) of the*  *-- affected accounts by adding sup-cost to their current values of details*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].addTransaction          @transactionNo  INT,          @cost           REAL,          @deptNo         INT,          @assembliesID   INT,          @processID      INT,          @jobNo          INT          AS          BEGIN  *-- Update the transaction table*              INSERT INTO [Project].[Transaction] VALUES (@transactionNo, @cost)  *-- Update the Relation table Records to associate with a job*              INSERT INTO [Project].[Records] VALUES (@jobNo, @transactionNo)              UPDATE  [Project].Assemblies              SET     [Project].Assemblies.details1 += @cost *-- Adds to the existing values in field*              WHERE   [Project].Assemblies.id = @assembliesID  *-- Update the Department Table for the given department*              UPDATE  [Project].Department              SET     [Project].Department.details2 += @cost *-- Adds to the existing values in field*              WHERE   [Project].Department.deptNo = @deptNo  *-- Update the Process table for the given Process*              UPDATE  [Project].Process              SET     [Project].Process.details3 += @cost *-- Adds to the existing values in field*              WHERE   [Project].Process.id = @processID          END      GO  *-----------------------------------------------------------------------------*  *-- 9. Retrieve the total cost incurred on an assembly-id*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].getTotalCosts          @id INT      AS          BEGIN              SELECT details1 AS TOTAL\_COST              FROM [Project].Assemblies  *-- Only show the selected assemblies ID*              WHERE id = @id          END      GO  *-----------------------------------------------------------------------------*  *-- 10. Retrieve the total labor time within a department for jobs completed in the*  *-- department during a given date*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].getTotalLaborTime          @deptNo  INT,          @endDate DATE      AS          BEGIN              SELECT                  deptNo,                  fit.timeOfLabor + cut.timeOfLabor + paint.timeOfLabor AS totalTimeOfLabor              FROM [Project].Job job  *-- Join job cut table*                  LEFT JOIN [Project].J\_Cut cut                      ON job.jobNo = cut.jobNo    *-- Join job fit table*                  LEFT JOIN [Project].J\_Fit fit                      ON job.jobNo = fit.jobNo    *-- Join job paint table*                  LEFT JOIN [Project].J\_Paint paint                      ON job.jobNo = paint.jobNo  *-- Only show the selected deptNo and end date*              WHERE                      deptNo  = @deptNo                  AND endDate = @endDate          END      GO  *-----------------------------------------------------------------------------*  *-- 11. Retrieve the processes through which a given assembly-id has passed so far*  *-- (in datecommenced order) and the department responsible for each process*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].getProcessUpdate          @assID INT      AS      BEGIN          SELECT              dateOrdered,              processID,              deptNo          FROM [Project].Process [p]  *-- Relation table between manufactures and assemblies*              LEFT JOIN [Project].Manufactures [m]                  ON p.id = m.processID  *-- Join assemblies table*              LEFT JOIN [Project].[Assemblies] [a]                  ON m.assembliesID = a.id  *-- Only show the given assemblies ID*          WHERE a.id = @assID          ORDER BY dateOrdered      END        GO  *-----------------------------------------------------------------------------*  *-- 12. Retrieve the jobs (together with their type information and assembly-id)*  *-- completed during a given date in a given department*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].getJobs              @deptNo  INT,              @endDate DATE      AS          BEGIN              SELECT                  job.jobNo,                  otherInfo,                  assembliesID              FROM [Project].Job job  *-- Join job cut table*                  LEFT JOIN [Project].J\_Cut cut                      ON job.jobNo = cut.jobNo    *-- Join job fit table*                  LEFT JOIN [Project].J\_Fit fit                      ON job.jobNo = fit.jobNo    *-- Join job paint table*                  LEFT JOIN [Project].J\_Paint paint                      ON job.jobNo = paint.jobNo              WHERE  *-- Conditional upon selected dept and date*                      deptNo  = @deptNo                  AND endDate = @endDate          END      GO  *-----------------------------------------------------------------------------*  *-- 13. Retrieve the customers (in name order) whose category is in a given range*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].getCustomers          @min INT,          @max INT      AS          BEGIN              SELECT [name]              FROM [Project].Customer  *-- Get the customers with category between min and max range*              WHERE category BETWEEN @min AND @max              ORDER BY [name]          END      GO  *-----------------------------------------------------------------------------*  *-- 14. Delete all cut-jobs whose job-no is in a given range*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].deleteJobs          @min INT,          @max INT      AS          BEGIN              DELETE FROM [Project].J\_Cut  *-- Delete cuts between min and max job number range*  *-- (with data related to a cut job)*              WHERE                      jobNo BETWEEN @min AND @max                  AND material IS NOT NULL          END      GO  *-----------------------------------------------------------------------------*  *-- 15. Change the color of a given paint job*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].setPaintJob          @newColor   VARCHAR(255),          @jobNo      INT      AS          BEGIN              UPDATE  [Project].J\_Paint              SET     [Project].J\_Paint.color = @newColor              WHERE   [Project].J\_Paint.jobNo = @jobNo          END      GO  *-----------------------------------------------------------------------------*  *-- 17. Retrieve the customers (in name order) whose category is in a given range*  *-----------------------------------------------------------------------------*      GO      CREATE PROCEDURE [Project].getCustomersInRange          @min INT,          @max INT      AS          BEGIN              SELECT \*              FROM [Project].Customer  *-- Only within the min and max range*              WHERE category BETWEEN @min AND @max              ORDER BY [name]          END      GO |

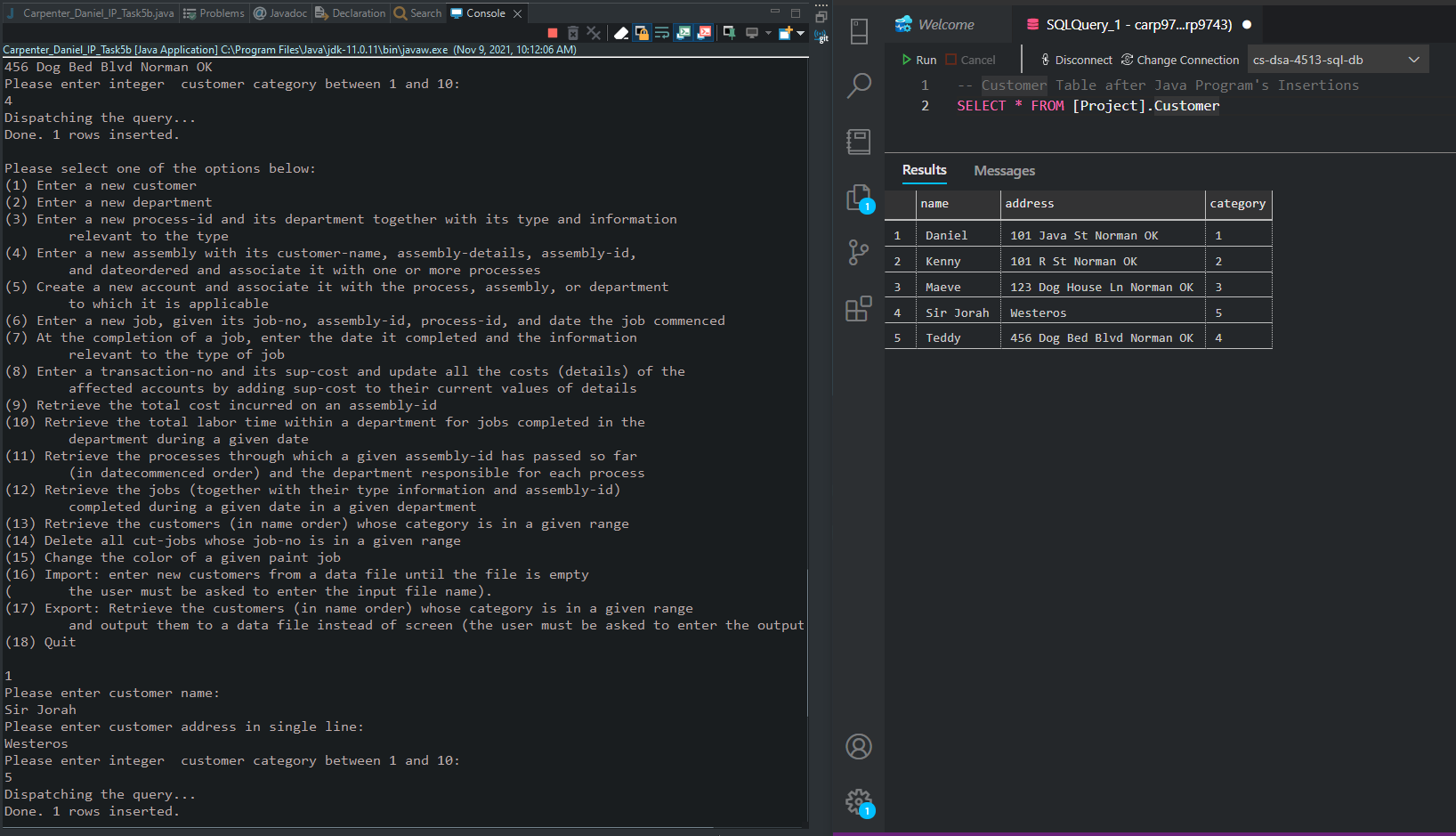
**5.2. Java Source Code**

|  |
| --- |
| **import** java.sql.CallableStatement;  **import** java.sql.Connection;  **import** java.util.Scanner;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.DriverManager;  **import** java.sql.PreparedStatement;  **import** java.io.\*;  **import** java.io.FileWriter;  // ==========================================================================  // @class: DSA 4513  // @asnmt: Class Project  // @task: 5 (b)  // @author: Daniel Carpenter, ID: 113009743  // @description:  // Program to implement job-shop accounting system queries  // ==========================================================================  **public** **class** Carpenter\_Daniel\_IP\_Task5b {  // Database credentials  **final** **static** String ***HOSTNAME*** = "carp9743.database.windows.net";  **final** **static** String ***DBNAME*** = "cs-dsa-4513-sql-db";  **final** **static** String ***USERNAME*** = "carp9743";  **final** **static** String ***PASSWORD*** = "tacoBout$97315!";  // Database connection string  **final** **static** String ***URL*** = String.*format*("jdbc:sqlserver://%s:1433;database=%s;user=%s;password=%s;encrypt=true;trustServerCertificate=false;hostNameInCertificate=\*.database.windows.net;loginTimeout=30;",  ***HOSTNAME***, ***DBNAME***, ***USERNAME***, ***PASSWORD***);    // Selected integer that quits application  **final** **static** **int** ***QUIT\_APPLICATION*** = 18;  // Create Template Queries to Execute stored procedures  **final** **static** String ***QUERY\_1*** = "EXEC [Project].addCustomer @name = ?, @address = ?, @category = ?";  **final** **static** String ***QUERY\_2*** = "EXEC [Project].addDepartment @deptNo = ?";  **final** **static** String ***QUERY\_3*** = "EXEC [Project].addProcess @id = ?, @deptNo = ?";  **final** **static** String ***QUERY\_4*** = "EXEC [Project].addAssembly @assID = ?, @dateOrdered = ?, @details = ?, @customerName = ?, @processID = ?";  **final** **static** String ***QUERY\_5*** = "EXEC [Project].addAccount @acctNo = ?, @dateEstablished = ?, @assembliesID = ?, @processID = ?, @deptNo = ?";  **final** **static** String ***QUERY\_6*** = "EXEC [Project].addJob @jobNo = ?, @startDate = ?, @assembliesID = ?, @processID = ?, @deptNo = ?";  **final** **static** String ***QUERY\_7*** = "EXEC [Project].setJobAsCompleted @jobNo = ?, @endDate = ?, @otherInfo = ?";  **final** **static** String ***QUERY\_8*** = "EXEC [Project].addTransaction @transactionNo = ?, @cost = ?, @deptNo = ?, @assembliesID = ?, @processID = ?, @jobNo = ?";  **final** **static** String ***QUERY\_9*** = "{CALL [Project].getTotalCosts(?)}";  **final** **static** String ***QUERY\_10*** = "{CALL [Project].getTotalLaborTime(?, ?)}";  **final** **static** String ***QUERY\_11*** = "{CALL [Project].getProcessUpdate(?)}";  **final** **static** String ***QUERY\_12*** = "{CALL [Project].getJobs(?, ?)}";  **final** **static** String ***QUERY\_13*** = "{CALL [Project].getCustomers(?, ?)}";  **final** **static** String ***QUERY\_14*** = "EXEC [Project].deleteJobs @min = ?, @max = ?";  **final** **static** String ***QUERY\_15*** = "EXEC [Project].setPaintJob @newColor = ?, @jobNo = ?";  **final** **static** String ***QUERY\_17*** = "{CALL [Project].getCustomersInRange(?, ?)}";    // User input prompt  **final** **static** String ***PROMPT*** =  "\nPlease select one of the options below: \n" +  "(1) Enter a new customer \n" +  "(2) Enter a new department \n" +  "(3) Enter a new process-id and its department together with its type and information \n" +  "\trelevant to the type\n" +  "(4) Enter a new assembly with its customer-name, assembly-details, assembly-id, \n" +  "\tand dateordered and associate it with one or more processes\n" +  "(5) Create a new account and associate it with the process, assembly, or department \n" +  "\tto which it is applicable\n" +  "(6) Enter a new job, given its job-no, assembly-id, process-id, and date the job commenced\n" +  "(7) At the completion of a job, enter the date it completed and the information \n" +  "\trelevant to the type of job \n" +  "(8) Enter a transaction-no and its sup-cost and update all the costs (details) of the \n" +  "\taffected accounts by adding sup-cost to their current values of details \n" +  "(9) Retrieve the total cost incurred on an assembly-id \n" +  "(10) Retrieve the total labor time within a department for jobs completed in the \n" +  "\tdepartment during a given date\n" +  "(11) Retrieve the processes through which a given assembly-id has passed so far \n" +  "\t(in datecommenced order) and the department responsible for each process\n" +  "(12) Retrieve the jobs (together with their type information and assembly-id) \n" +  "\tcompleted during a given date in a given department\n" +  "(13) Retrieve the customers (in name order) whose category is in a given range\n" +  "(14) Delete all cut-jobs whose job-no is in a given range\n" +  "(15) Change the color of a given paint job\n" +  "(16) Import: enter new customers from a data file until the file is empty \n" +  "(\tthe user must be asked to enter the input file name). \n" +  "(17) Export: Retrieve the customers (in name order) whose category is in a given range \n" +  "\tand output them to a data file instead of screen (the user must be asked to enter the output file name).\n" +  "(18) Quit\n";    // Function to read in a csv file and return a concatenated into an insert statement  **public** **static** String readCSV(String filename) **throws** IOException, SQLException {  // Number of columns in the customer table (3)  **final** **int** NUM\_CUST\_COLS = 3;    // string that will hold the insert statement  String insertStatement = "INSERT INTO [Project].Customer VALUES (";    // Create input reader  BufferedReader input = **new** BufferedReader(**new** FileReader(filename));  String line = "";  **int** iterCount = 0; // keep track of iterations  **final** **int** FIRST\_ITER = 0;    // Iterate through each 'row' of the csv  **while** ((line = input.readLine()) != **null**) {    // IF the first iteration, then do nothing. else concatenate parenthesis  **if** (iterCount != FIRST\_ITER) {  insertStatement += ", (";  } **else** {  ++iterCount;  }      // Iterate through each 'column' of the csv file  **for** (**int** col = 0; col < NUM\_CUST\_COLS; ++col) {    // Add a ' in front of the string vars  **if** (col != NUM\_CUST\_COLS - 1) {  insertStatement += "'";  }    // return the value of the row for each column index (1 through 3)  insertStatement += line.split(",")[col];    // If not at last column, add comma to the string  // Add a ' at end of the string vars  **if** (col != NUM\_CUST\_COLS - 1) {  insertStatement += "', ";  }    // End the values insert  **else** {  insertStatement += ")";  }  }  }    // close the input method  input.close();    // return the insert statement  **return** (insertStatement);  }  **public** **static** **void** main(String[] args) **throws** SQLException, IOException {  System.***out***.println("Update Job-Shop Accounting Database:");  // CREATE INPUT SCANNER ---------------------------------------------------------------------------------  **final** Scanner sc = **new** Scanner(System.***in***); // Scanner is used to collect thes user input  String option = ""; // Initialize user option selection as nothing  **while** (!option.equals(Integer.*toString*(***QUIT\_APPLICATION***))) { // As user for options until quit option is selected  System.***out***.println(***PROMPT***); // Print the available options    option = sc.next(); // Read in the user option selection    // BEGIN SWITCH STATEMENTS ==========================================================================  **switch** (option) { // Switch between different options    // (1) Enter a new customer  **case** "1":    // Set the query  String query = ***QUERY\_1***;    System.***out***.println("Please enter customer name:");  sc.nextLine();  String cName = sc.nextLine();    System.***out***.println("Please enter customer address in single line:");  String address = sc.nextLine();    System.***out***.println("Please enter integer customer category between 1 and 10:");  **int** category = sc.nextInt();  // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setString(1, cName);  statement.setString(2, address);  statement.setInt(3, category);  System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (2) Enter a new department  **case** "2":    // Set the query  query = ***QUERY\_2***;    System.***out***.println("Please enter a new department number:");  **int** deptNo = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, deptNo);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (3) Enter a new process-id and its department together with its type and information  // relevant to the type  **case** "3":    // Set the query  query = ***QUERY\_3***;    System.***out***.println("Please enter a new process id (integer):");  **int** processID = sc.nextInt();    System.***out***.println("Please enter its existing department number:");  deptNo = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, processID);  statement.setInt(2, deptNo);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (4) Enter a new assembly with its customer-name, assembly-details, assembly-id,  // and dateordered and associate it with one or more processes  **case** "4":    // Set the query  query = ***QUERY\_4***;    System.***out***.println("Please enter a new assembly id (integer):");  **int** assID = sc.nextInt();    System.***out***.println("Please enter the date commenced in 'YYYY-MM-DD' format, e.g. 2020-11-30:");  sc.nextLine();  String dateOrdered = sc.nextLine();    System.***out***.println("Please enter the details of the order in text");  String details = sc.nextLine();    System.***out***.println("Please enter the existing customer name associated with the assembly");  cName = sc.nextLine();    System.***out***.println("Please enter the existing process ID (integer) associated with the assembly");  processID = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, assID);  statement.setString(2, dateOrdered);  statement.setString(3, details);  statement.setString(4, cName);  statement.setInt(5, processID);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (5) Create a new account and associate it with the process, assembly, or department  // to which it is applicable  **case** "5":    // Set the query  query = ***QUERY\_5***;    System.***out***.println("Please enter a new account id (integer):");  **int** acctNo = sc.nextInt();    System.***out***.println("Please enter the date established in 'YYYY-MM-DD' format, e.g. 2020-11-30:");  sc.nextLine();  String dateEst = sc.nextLine();    System.***out***.println("Please enter the associated assembly id (integer)");  assID = sc.nextInt();    System.***out***.println("Please enter the associated process id (integer)");  processID = sc.nextInt();    System.***out***.println("Please enter the associated department number");  deptNo = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, acctNo);  statement.setString(2, dateEst);  statement.setInt(3, assID);  statement.setInt(4, processID);  statement.setInt(5, deptNo);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (6) Enter a new job, given its job-no, assembly-id, process-id, and date the job commenced  **case** "6":    // Set the query  query = ***QUERY\_6***;    System.***out***.println("Please enter a new job number:");  **int** jobNo = sc.nextInt();    System.***out***.println("Please enter the start date of the job in 'YYYY-MM-DD' format, e.g. 2020-11-30:");  sc.nextLine();  String startDate = sc.nextLine();    System.***out***.println("Please enter the associated assembly id (integer)");  assID = sc.nextInt();    System.***out***.println("Please enter the associated process id (integer)");  processID = sc.nextInt();    System.***out***.println("Please enter the associated department number");  deptNo = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, jobNo);  statement.setString(2, startDate);  statement.setInt(3, assID);  statement.setInt(4, processID);  statement.setInt(5, deptNo);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (7) At the completion of a job, enter the date it completed and the information  // relevant to the type of job  **case** "7":    // Set the query  query = ***QUERY\_7***;    System.***out***.println("Please enter the existing job number:");  jobNo = sc.nextInt();    System.***out***.println("Please enter the end date of the job in 'YYYY-MM-DD' format, e.g. 2020-11-30:");  sc.nextLine();  String endDate = sc.nextLine();    System.***out***.println("Please enter any other info about the job (as text):");  String otherInfo = sc.nextLine();  // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, jobNo);  statement.setString(2, endDate);  statement.setString(3, otherInfo);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (8) Enter a transaction-no and its sup-cost and update all the costs (details) of the  // affected accounts by adding sup-cost to their current values of details  **case** "8":    // Set the query  query = ***QUERY\_8***;    System.***out***.println("Please enter the new transaction number:");  **int** transactionNo = sc.nextInt();    System.***out***.println("Please enter the cost of the transaction (as decimal number):");  **double** cost = sc.nextDouble();    System.***out***.println("Please enter the associated assembly id (integer)");  assID = sc.nextInt();    System.***out***.println("Please enter the associated process id (integer)");  processID = sc.nextInt();    System.***out***.println("Please enter the associated department number");  deptNo = sc.nextInt();    System.***out***.println("Please enter the associated job number");  jobNo = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Activate stored procedure to enter above data in database  // Insert new performer record into database  **try** (  **final** PreparedStatement statement = connection.prepareStatement(query)) {  // Populate the query template with the data collected from the user  statement.setInt(1, transactionNo);  statement.setDouble(2, cost);  statement.setInt(3, deptNo);  statement.setInt(4, assID);  statement.setInt(5, processID);  statement.setInt(6, jobNo);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = statement.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  }  **break**;    // (9) Retrieve the total cost incurred on an assembly-id  **case** "9":    // Set the query  query = ***QUERY\_9***;    System.***out***.println("Please enter the assembly id (integer):");  assID = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  CallableStatement cs = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  cs.setInt("id", assID);    // Run the stored procedure and store values in resultSet  System.***out***.println("Dispatching the query...");  ResultSet resultSet = cs.executeQuery();    System.***out***.println("Done.");  System.***out***.println("\nTotal cost incurred on assembly-id: " + assID);  // Unpack the tuples returned by the database and print them out to the user  **while** (resultSet.next()) {  System.***out***.println(String.*format*("%s", resultSet.getString(1)));  }  }    **break**;    // (10) Retrieve the total labor time within a department for jobs completed in the  // department during a given date  **case** "10":    // Set the query  query = ***QUERY\_10***;    System.***out***.println("Please enter the department number:");  deptNo = sc.nextInt();    System.***out***.println("Please enter the end date of the job in 'YYYY-MM-DD' format, e.g. 2020-11-30:");  sc.nextLine();  endDate = sc.nextLine();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  CallableStatement cs = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  cs.setInt("deptNo", deptNo);  cs.setString("endDate", endDate);    // Run the stored procedure and store values in resultSet  System.***out***.println("Dispatching the query...");  ResultSet resultSet = cs.executeQuery();    System.***out***.println("Done.");  System.***out***.println("\nTotal labor time for department: " + deptNo +  " for date ending on: " + endDate);  System.***out***.println("deptNo | timeOfLabor");    // Unpack the tuples returned by the database and print them out to the user  **while** (resultSet.next()) {  System.***out***.println(String.*format*("%s | %s",  resultSet.getString(1),  resultSet.getString(2)));  }  }    **break**;      // (11) Retrieve the processes through which a given assembly-id has passed so far  // (in datecommenced order) and the department responsible for each process  **case** "11":    // Set the query  query = ***QUERY\_11***;    System.***out***.println("Please enter the assembly id (integer):");  assID = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  CallableStatement cs = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  cs.setInt("assID", assID);    // Run the stored procedure and store values in resultSet  System.***out***.println("Dispatching the query...");  ResultSet resultSet = cs.executeQuery();    System.***out***.println("Done.");  System.***out***.println("\nProcess for assembly-id: " + assID +  ", and its departement number; Sorted by date commenced.");  System.***out***.println("dateOrdered | processID | deptNo");    // Unpack the tuples returned by the database and print them out to the user  **while** (resultSet.next()) {  System.***out***.println(String.*format*("%s | %s | %s",  resultSet.getString(1),  resultSet.getString(2),  resultSet.getString(3)));  }  }    **break**;    // (12) Retrieve the jobs (together with their type information and assembly-id)  // completed during a given date in a given department  **case** "12":    // Set the query  query = ***QUERY\_12***;    System.***out***.println("Please enter the department number:");  deptNo = sc.nextInt();    System.***out***.println("Please enter the end date of the job in 'YYYY-MM-DD' format, e.g. 2020-11-30:");  sc.nextLine();  endDate = sc.nextLine();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  CallableStatement cs = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  cs.setInt("deptNo", deptNo);  cs.setString("endDate", endDate);    // Run the stored procedure and store values in resultSet  System.***out***.println("Dispatching the query...");  ResultSet resultSet = cs.executeQuery();    System.***out***.println("Done.");  System.***out***.println("\nJobs from department " + deptNo +  " completed on: " + endDate);  System.***out***.println("jobNo | otherInfo | assembliesID ");    // Unpack the tuples returned by the database and print them out to the user  **while** (resultSet.next()) {  System.***out***.println(String.*format*("%s | %s | %s",  resultSet.getString(1),  resultSet.getString(2),  resultSet.getString(3)));  }  }    **break**;    // (13) Retrieve the customers (in name order) whose category is in a given range  **case** "13":    // Set the query  query = ***QUERY\_13***;    System.***out***.println("Please enter MIN category number (integer from 1 - 10, inclusive):");  **int** min = sc.nextInt();    System.***out***.println("Please enter MAX category number (integer from 1 - 10, inclusive):");  **int** max = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  CallableStatement cs = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  cs.setInt("min", min);  cs.setInt("max", max);    // Run the stored procedure and store values in resultSet  System.***out***.println("Dispatching the query...");  ResultSet resultSet = cs.executeQuery();    System.***out***.println("Done.");  System.***out***.println("\nCustomers with category from " + min + " to " + max);  System.***out***.println("name"); // | otherInfo | assembliesID ");    // Unpack the tuples returned by the database and print them out to the user  **while** (resultSet.next()) {  System.***out***.println(String.*format*("%s", // | %s | %s",  resultSet.getString(1)));  }  }    **break**;    // (14) Delete all cut-jobs whose job-no is in a given range  **case** "14":    // Set the query  query = ***QUERY\_14***;    System.***out***.println("Please enter MIN category number (integer from 1 - 10, inclusive):");  min = sc.nextInt();    System.***out***.println("Please enter MAX category number (integer from 1 - 10, inclusive):");  max = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  PreparedStatement ps = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  ps.setInt(1, min);  ps.setInt(2, max);        System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_deleted = ps.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows deleted.", rows\_deleted) +  " from " + min + " to " + max);  }    **break**;    // (15) Change the color of a given paint job  **case** "15":    // Set the query  query = ***QUERY\_15***;    System.***out***.println("Please enter the new color:");  sc.nextLine();  String newColor = sc.nextLine();    System.***out***.println("Please enter the job number associated:");  jobNo = sc.nextInt();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  PreparedStatement ps = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  ps.setString(1, newColor);  ps.setInt(2, jobNo);        System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_changed = ps.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows changed.", rows\_changed) +  " for job number: " + jobNo);  }    **break**;    // (16) Import: enter new customers from a data file until the file is empty  // (the user must be asked to enter the input file name).    **case** "16":    System.***out***.println("Please enter the location and name of a CSV file with customer data:" +  "\n>> PLEASE DO NOT INCLUDE COMMAS EXCEPT FOR THE DELIMITER <<");  sc.nextLine();  String filename = sc.nextLine();    // create insert statement with values from csv file  query = *readCSV*(filename);      // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  PreparedStatement ps = connection.prepareCall(query);    System.***out***.println("Dispatching the query...");  // Actually execute the populated query  **final** **int** rows\_inserted = ps.executeUpdate();  System.***out***.println(String.*format*("Done. %d rows inserted.", rows\_inserted));  }  **break**;      // (17) Export: Retrieve the customers (in name order) whose category is in a given range  // and output them to a data file instead of screen (the user must be asked to enter the output file name).    **case** "17":    // Set the query  query = ***QUERY\_17***;    System.***out***.println("Please enter MIN category number (integer from 1 - 10, inclusive):");  min = sc.nextInt();    System.***out***.println("Please enter MAX category number (integer from 1 - 10, inclusive):");  max = sc.nextInt();    System.***out***.println("Please enter the file output name:");  sc.nextLine();  filename = sc.nextLine();    // Get a database connection and prepare a query statement  **try** (**final** Connection connection = DriverManager.*getConnection*(***URL***)) {    // Prepare a call to the stored procedure  CallableStatement cs = connection.prepareCall(query);    // Set the assigned value(s) to the procedures input  cs.setInt("min", min);  cs.setInt("max", max);    // Run the stored procedure and store values in resultSet  System.***out***.println("Dispatching the query...");  ResultSet resultSet = cs.executeQuery();    **try** {  FileWriter myWriter = **new** FileWriter(filename + ".csv");  myWriter.write("name,address,category\n");    // Unpack the tuples returned by the database and print them out to the user  **while** (resultSet.next()) {  myWriter.write(String.*format*("%s,%s,%s\n",  resultSet.getString(1),  resultSet.getString(2),  resultSet.getString(3)));  }    // close the writer  myWriter.close();    } **catch** (IOException e) {  System.***out***.println("Error with file name.");  e.printStackTrace();  }  }    System.***out***.println("Done. File Location here:");  System.***out***.println(filename + ".csv");    **break**;    // (18) Quit  **case** "18":  System.***out***.println("Finished! Your work here is done.");  }  }  sc.close(); // Close the scanner before exiting the application  }  } |

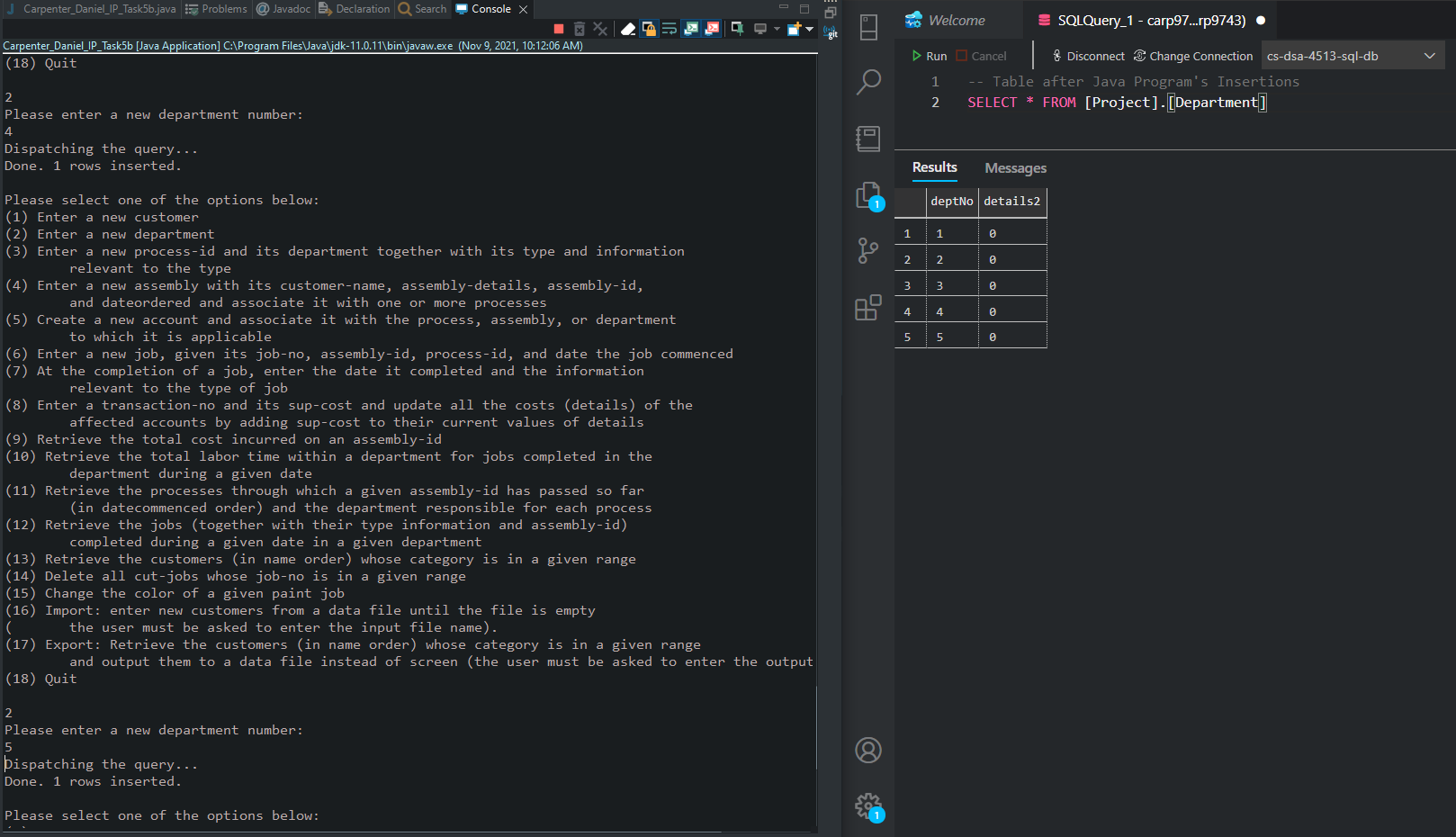
**5.2. Continued: Java Program Screenshot of Successful Compilation**

****

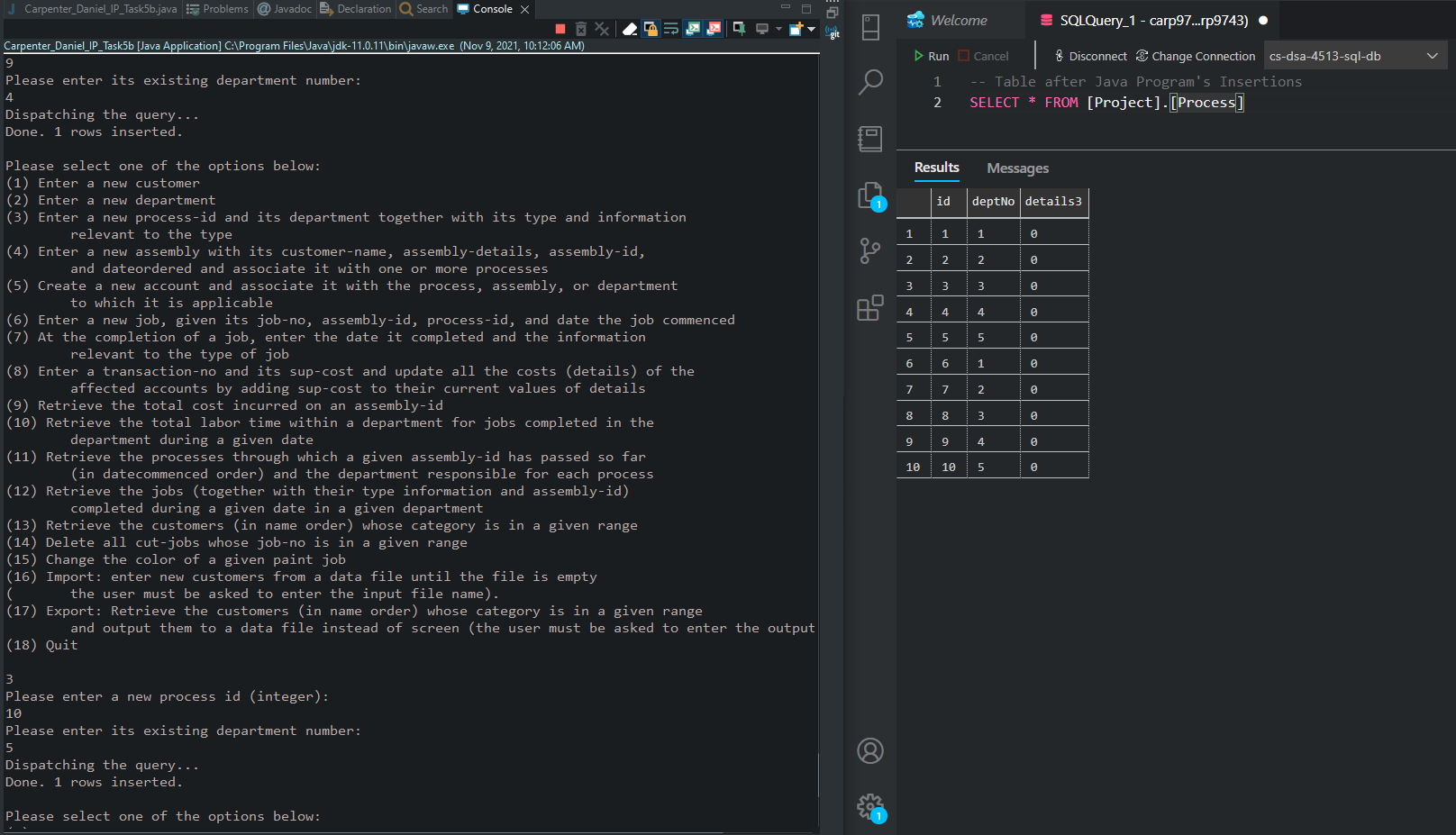
**6.1. Screenshots showing the testing of Query 1**

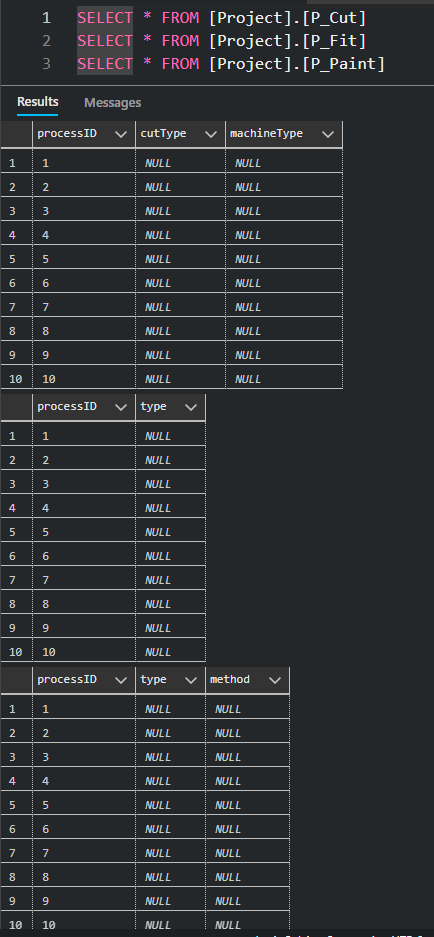
****

**6.2. Screenshots showing the testing of Query 2**

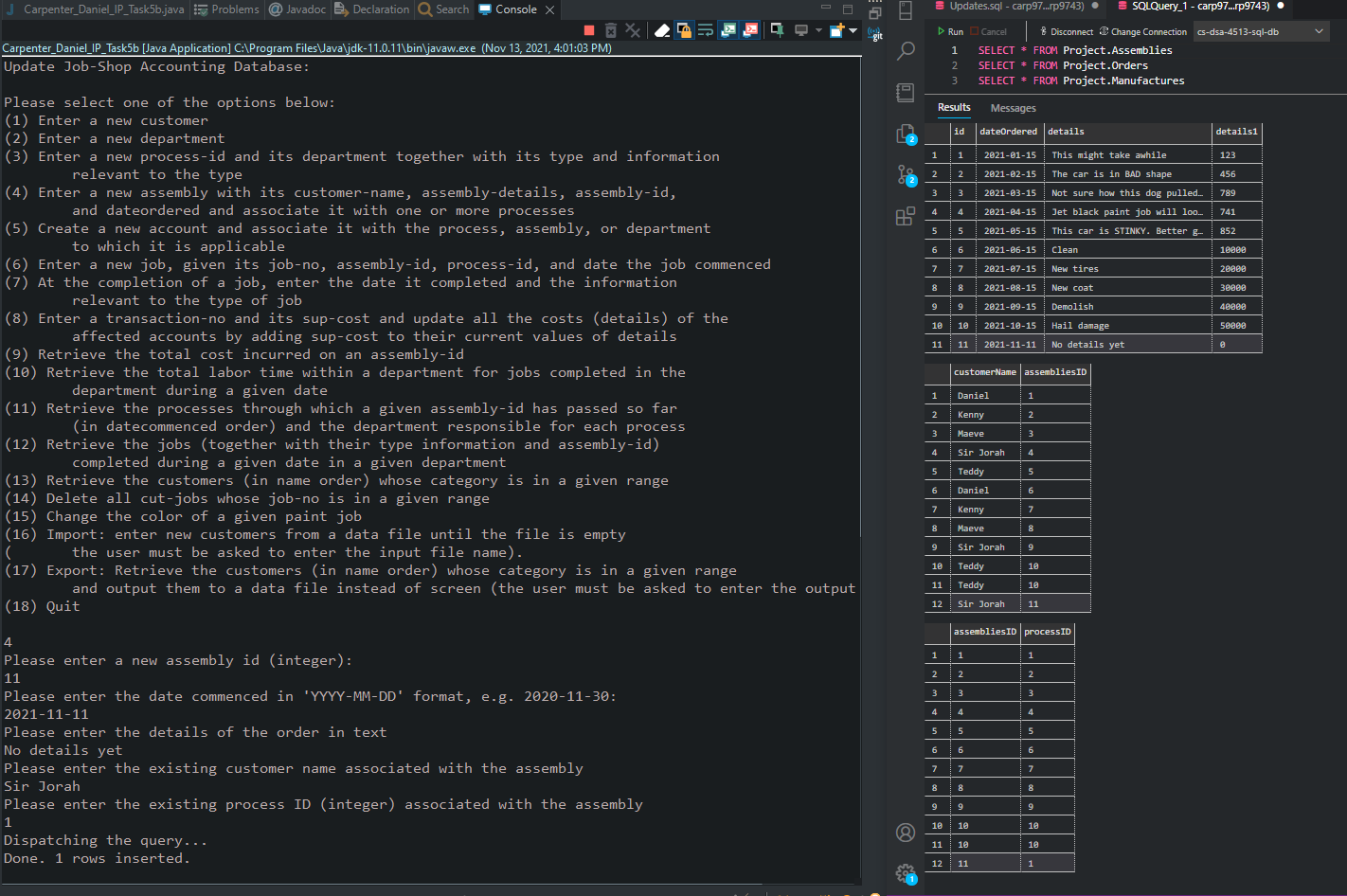
****

**6.3. Screenshots showing the testing of Query 3**

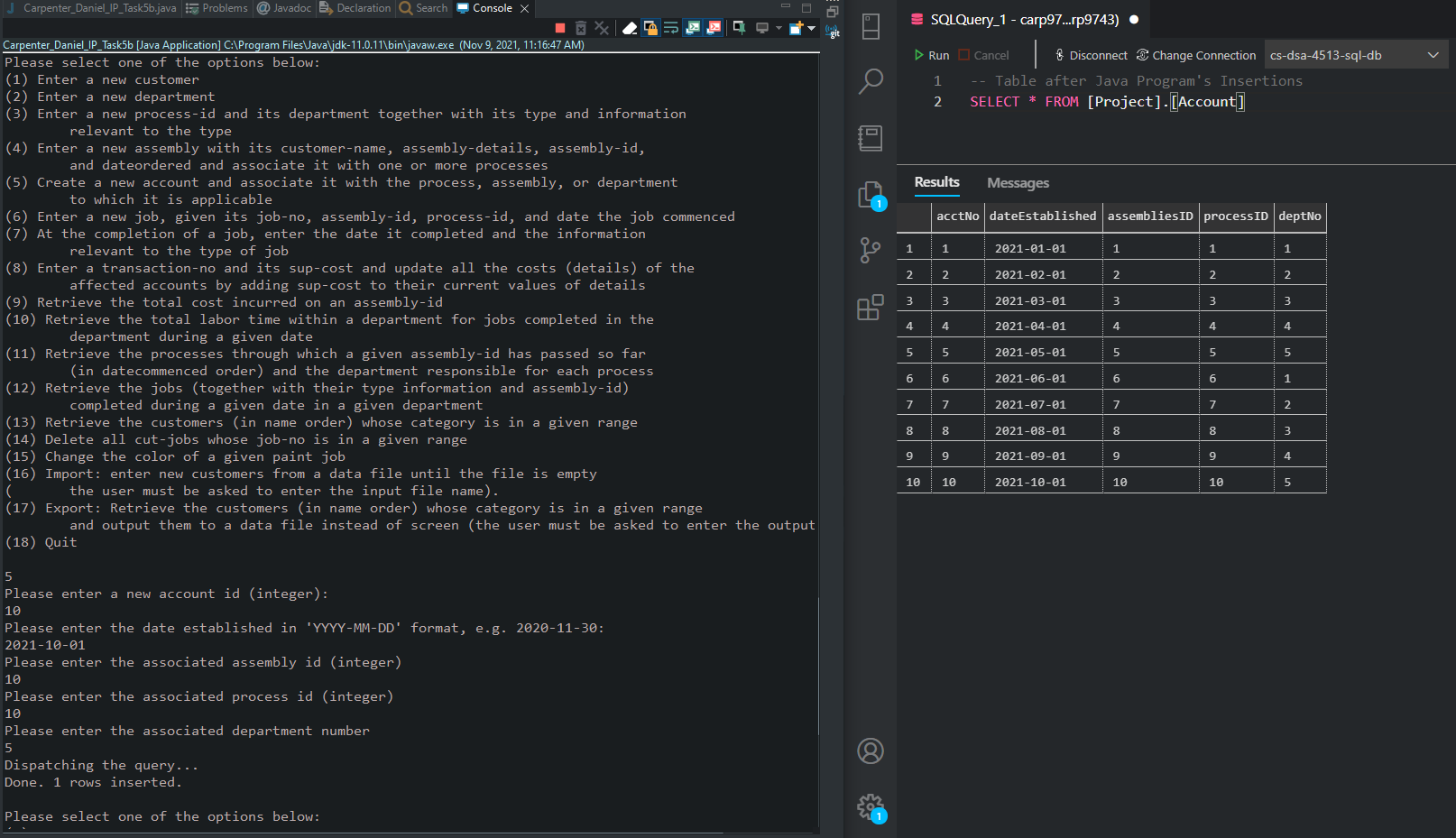
****

**6.3. Screenshots showing the testing of Query 3 *(Continued)***

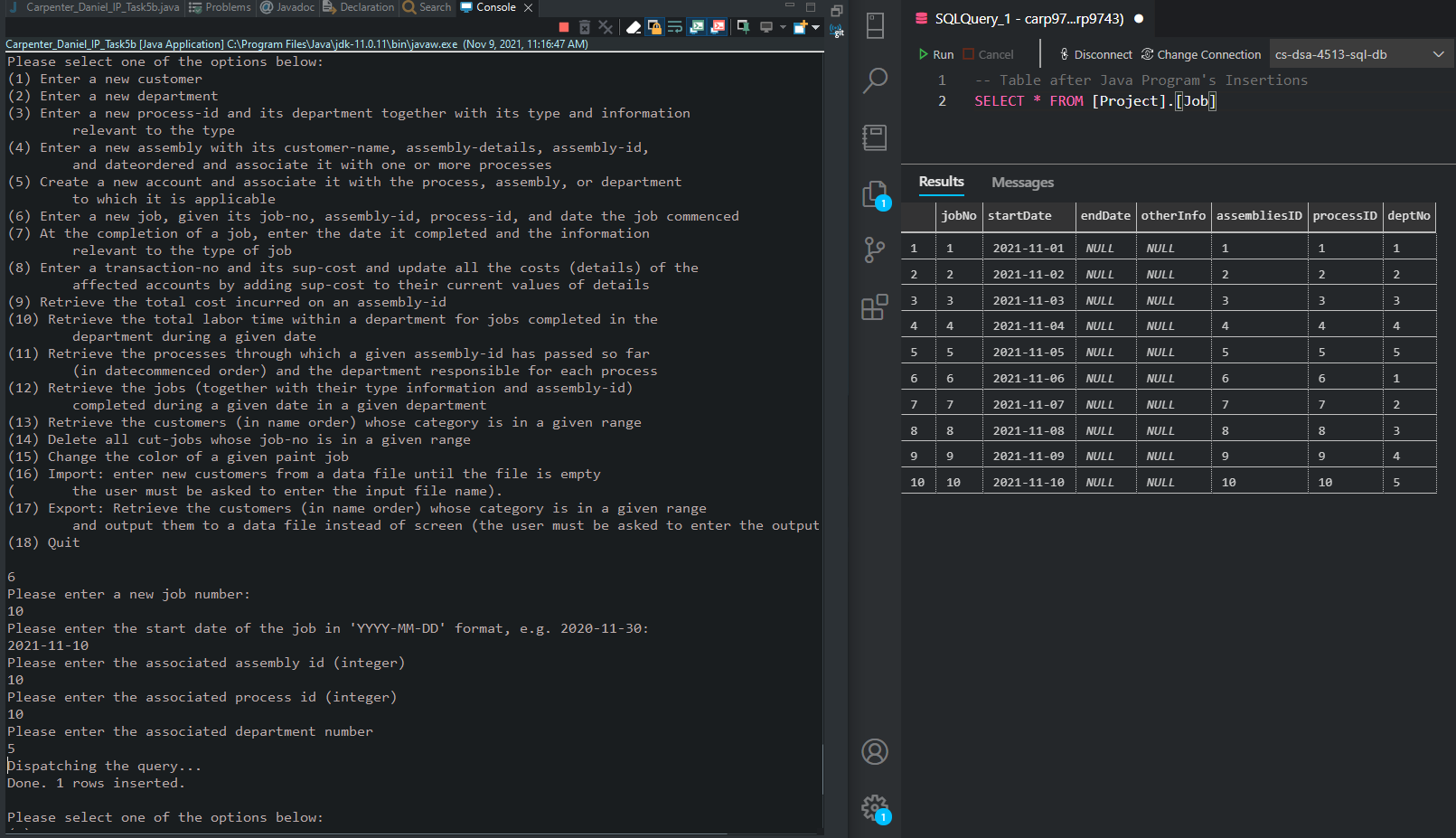
**6.4. Screenshots showing the testing of Query 4**

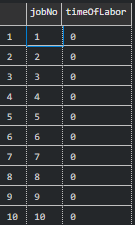
****

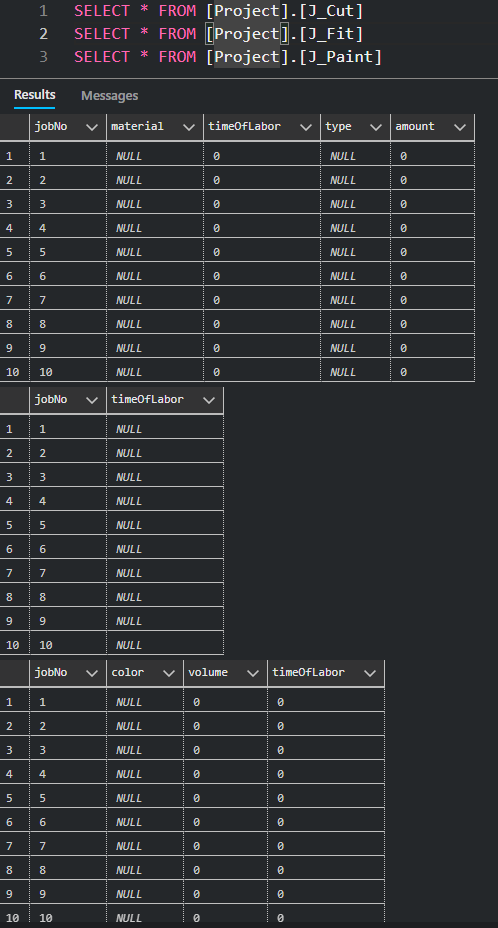
**6.5. Screenshots showing the testing of Query 5**

****

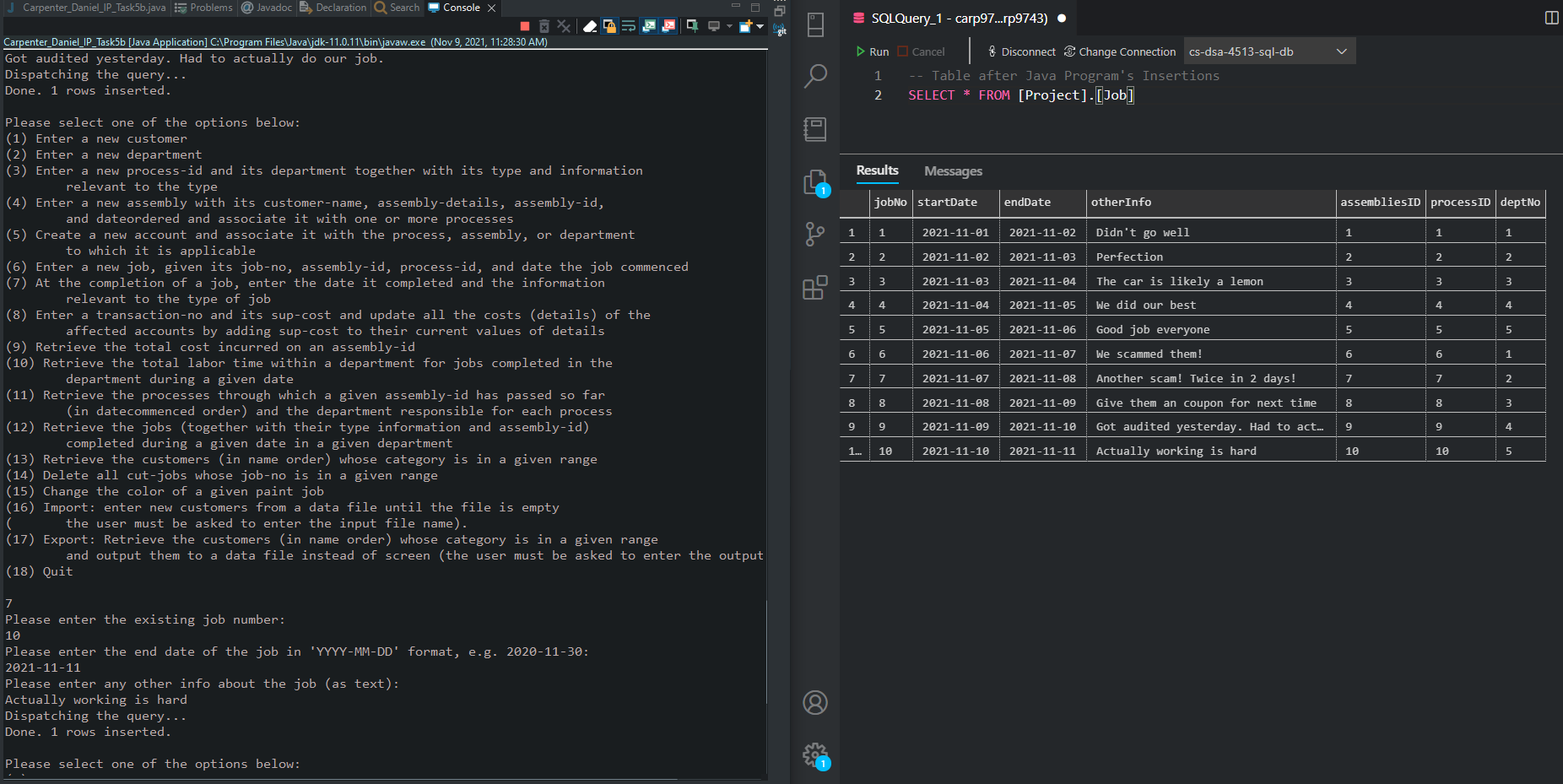
**6.6. Screenshots showing the testing of Query 6**

****

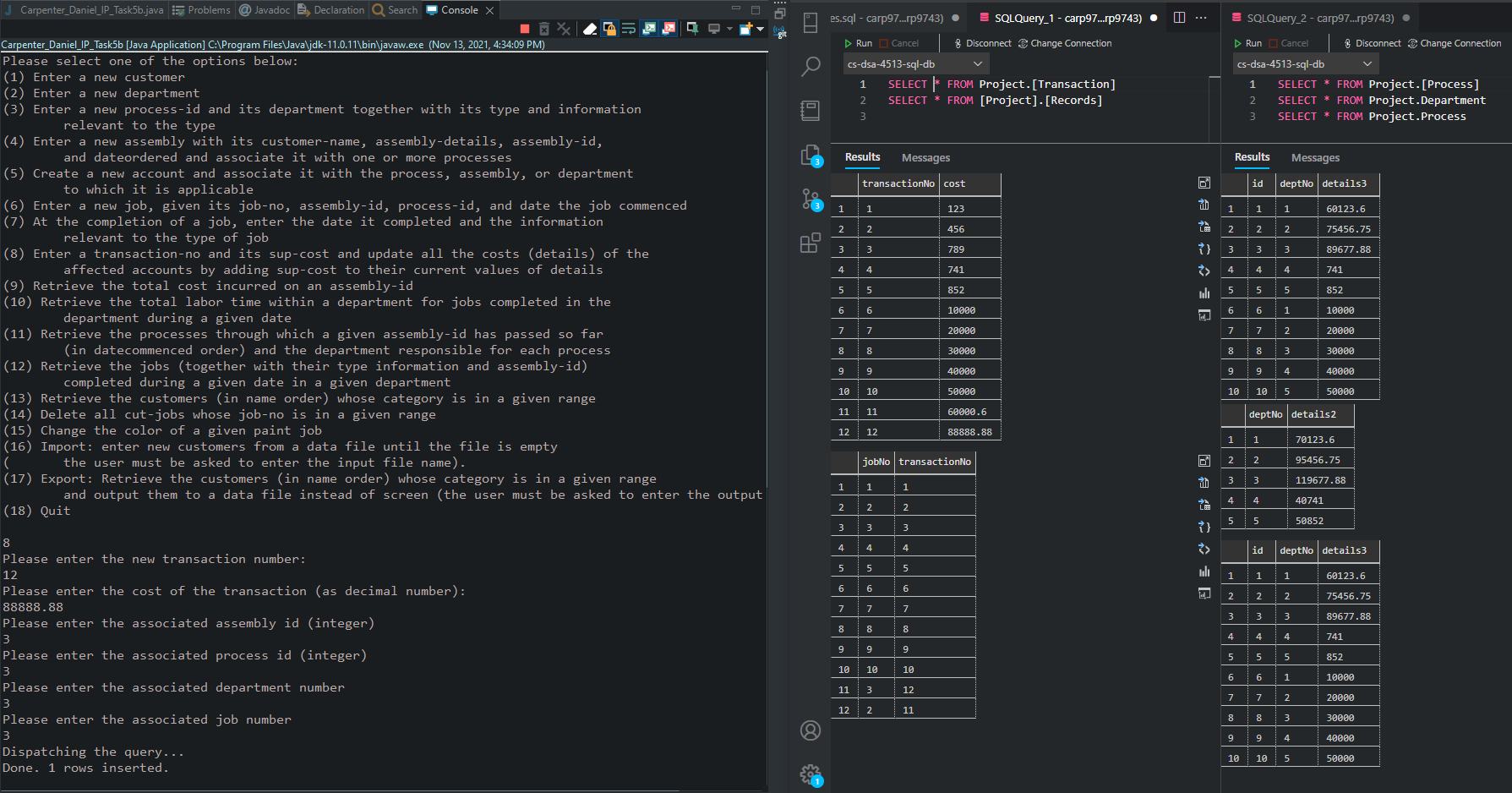
**6.6. Screenshots showing the testing** of **Query 6 *(Continued)***

****

**6.7. Screenshots showing the testing of Query 7**

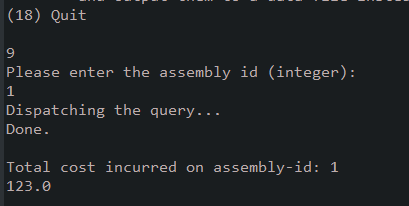
****

**6.8. Screenshots showing the testing of Query 8**

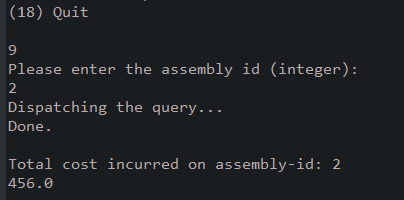
****

**6.9. Screenshots showing the testing of Query 9**

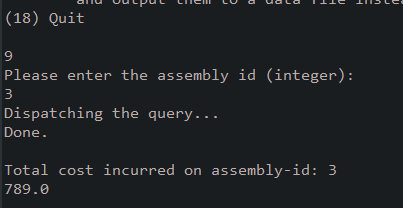
**Example 1**

****

**Example 2**

****

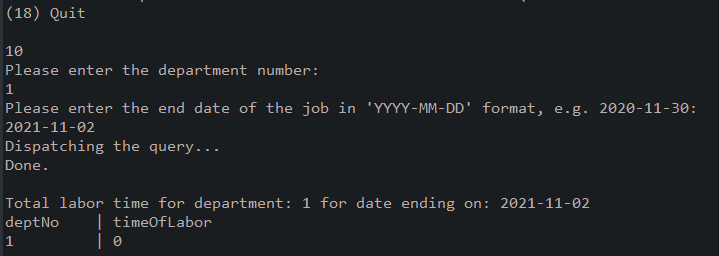
**Example 3**

****

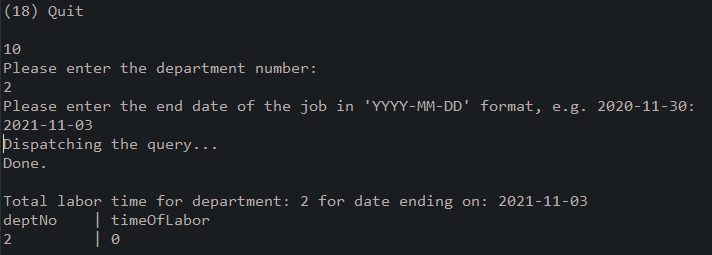
**6.10. Screenshots showing the testing of Query 10**

*Please note that these values are 0 since no values of `timeOfLabor` have been supplied in the database yet.*

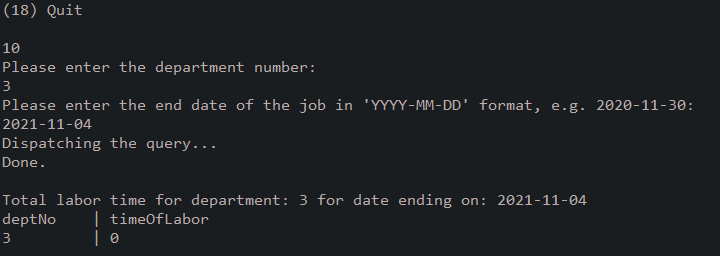
**Example 1**

****

**Example 2**

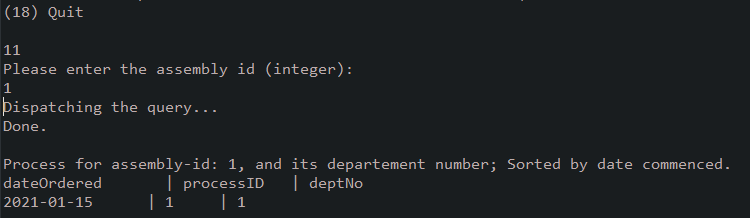
****

**Example 3**

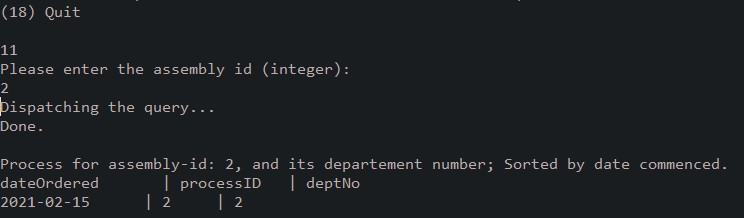
****

**6.11. Screenshots showing the testing of Query 11**

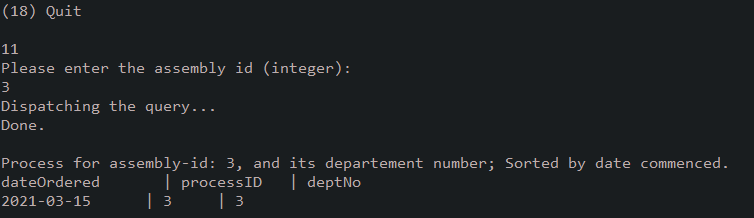
**Example 1**

****

**Example 2**

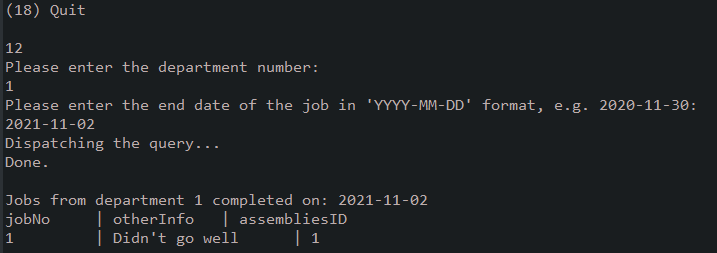
****

**Example 3**

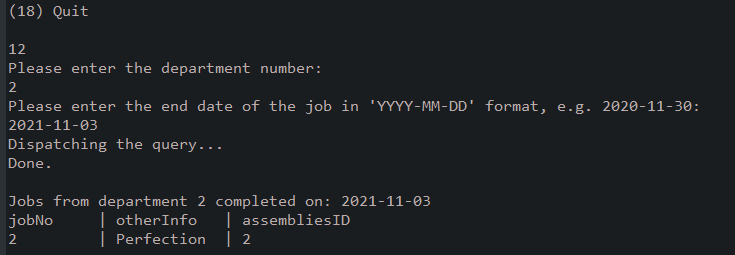
****

**6.12. Screenshots showing the testing of Query 12**

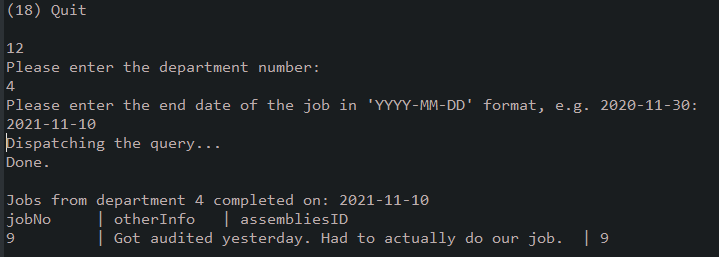
**Example 1**

****

**Example 2**

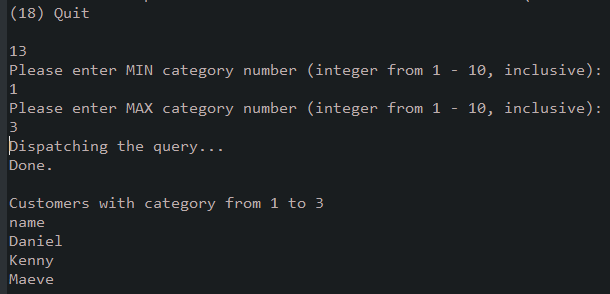
****

**Example 3**

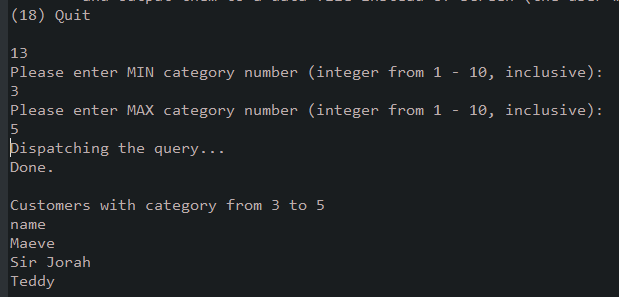
****

**6.13. Screenshots showing the testing of Query 13**

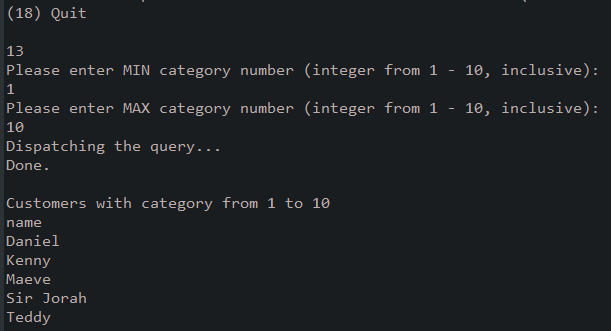
**Example 1**

****

**Example 2**

****

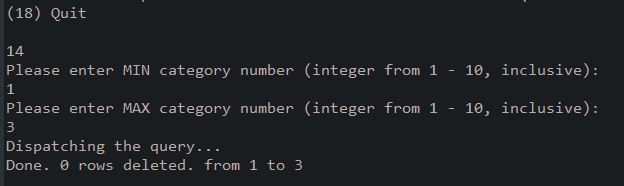
**Example 3**

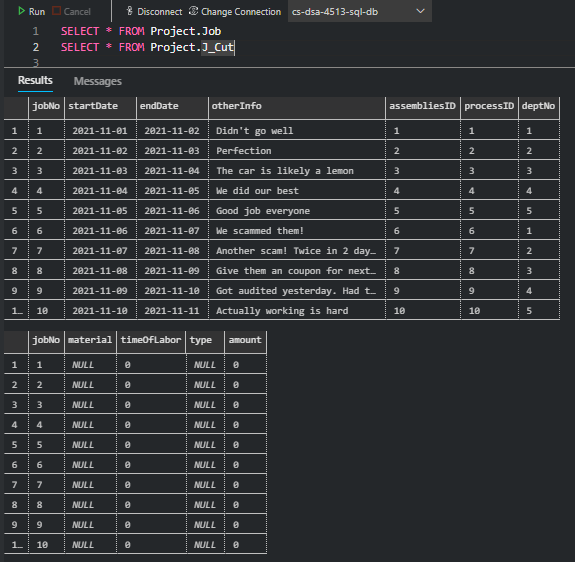
****

**6.14. Screenshots showing the testing of Query 14**

*None deleted since no values of J\_Cut have been supplied that specify if the job will be cut. I created the procedure this way to avoid deleting rows when not needed. If another procedure were to specify that the job was a cut job, then it would make more sense.*

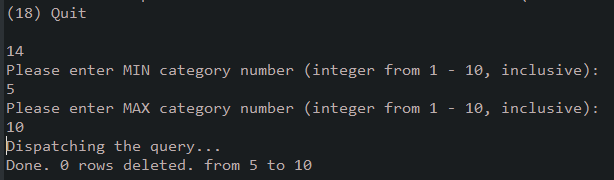
**Example 1**

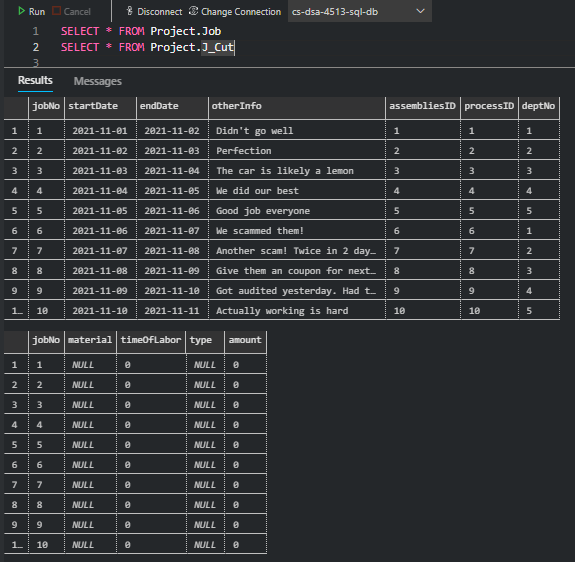
****

****

**6.14. Screenshots showing the testing of Query 14 *(Continued)***

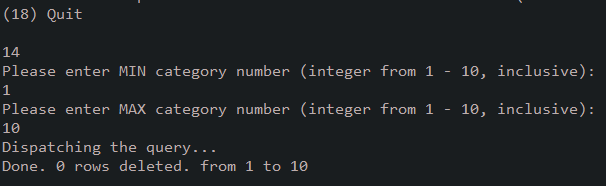
**Example 2**

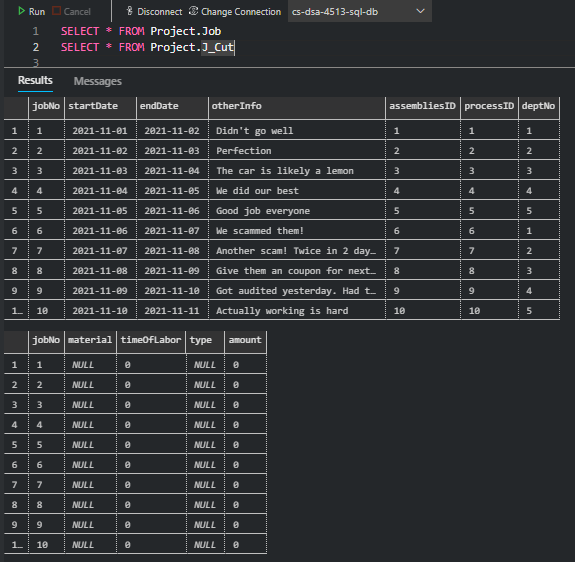
****

****

**6.14. Screenshots showing the testing of Query 14 *(Continued)***

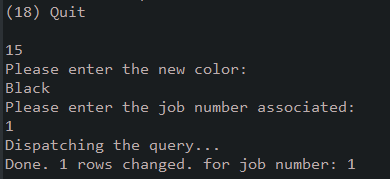
**Example 3**

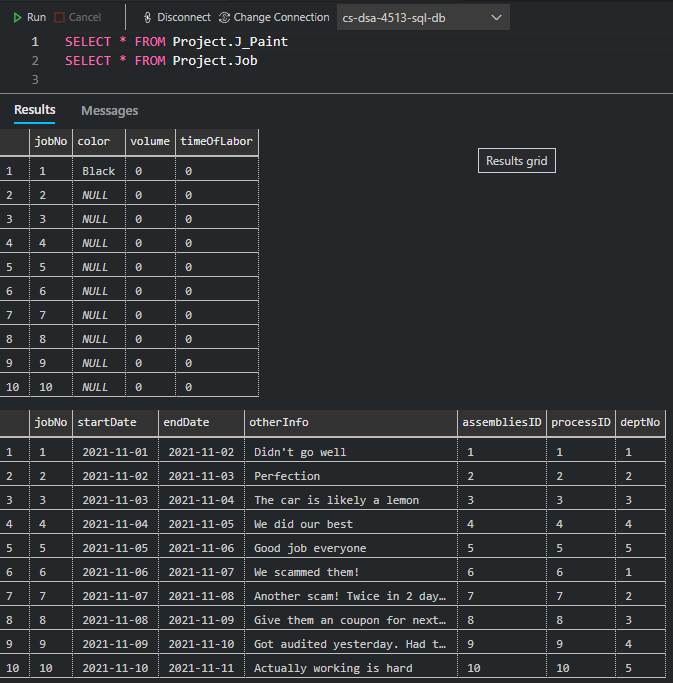
****

****

**6.15. Screenshots showing the testing of Query 15**

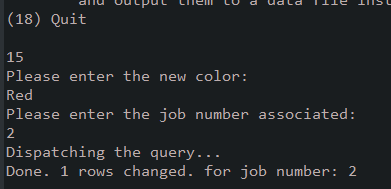
**Example 1**

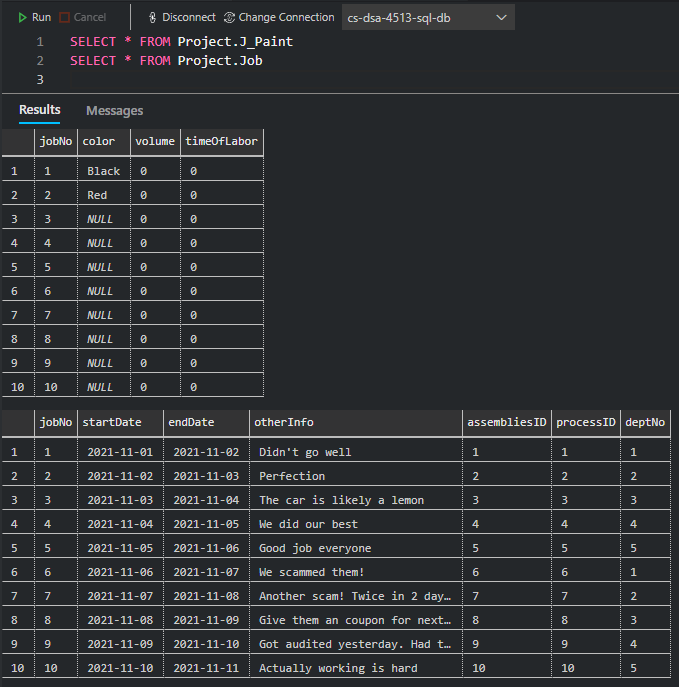
****

****

**6.15. Screenshots showing the testing of Query 15 *(Continued)***

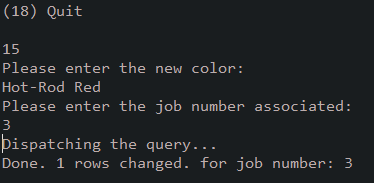
**Example 2**

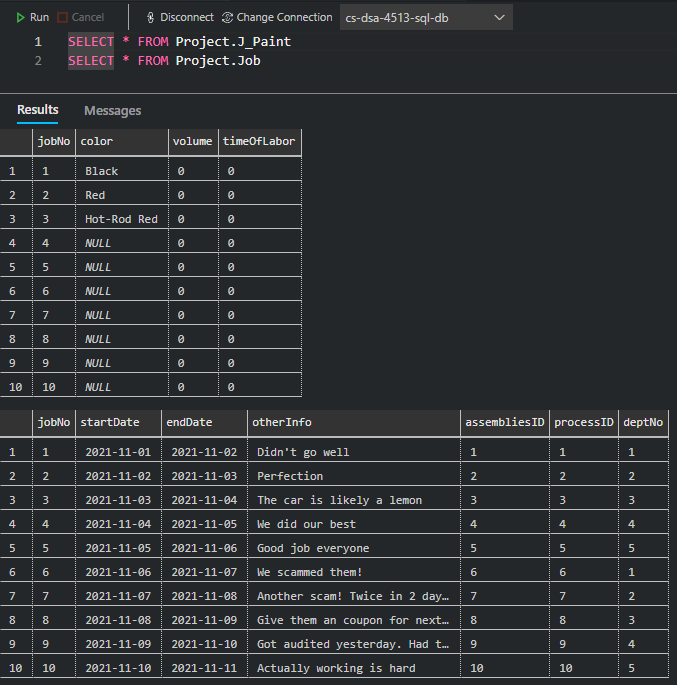
****

****

**6.15. Screenshots showing the testing of Query 15 *(Continued)***

**Example 3**

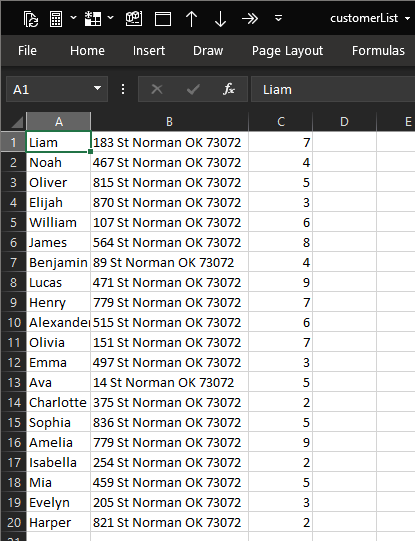
****

****

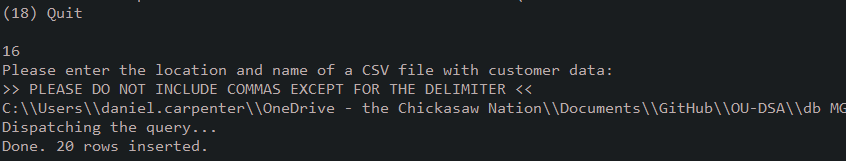
**6.16. Screenshots Showing the Testing of the Import and Export Options**

**Import Option (16):**

*customerList.csv contents:*

**

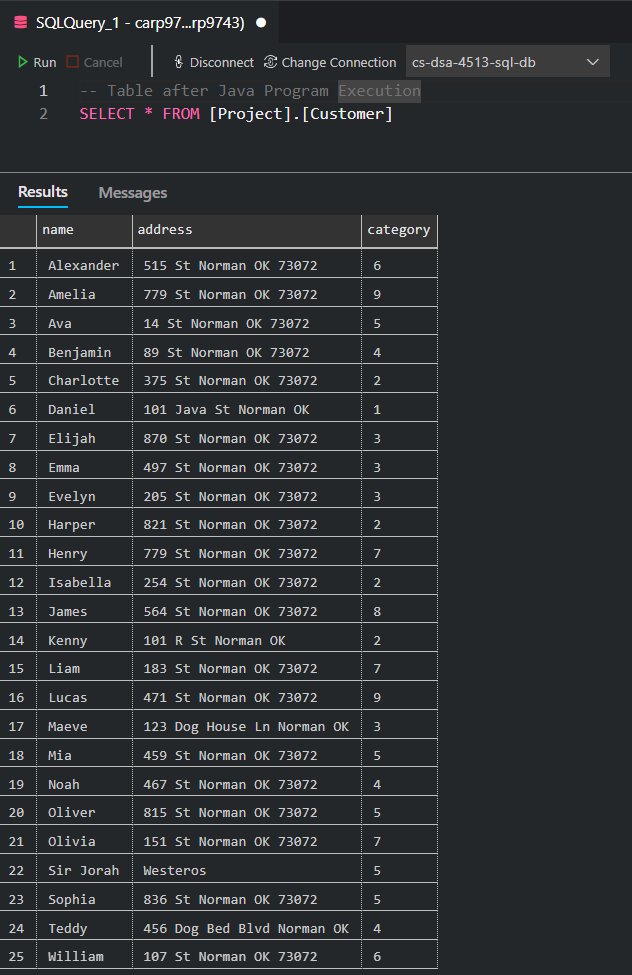
*Input file name and path to upload to database:*

****

**6.16. Screenshots showing the testing of the import and export options *(Continued)***

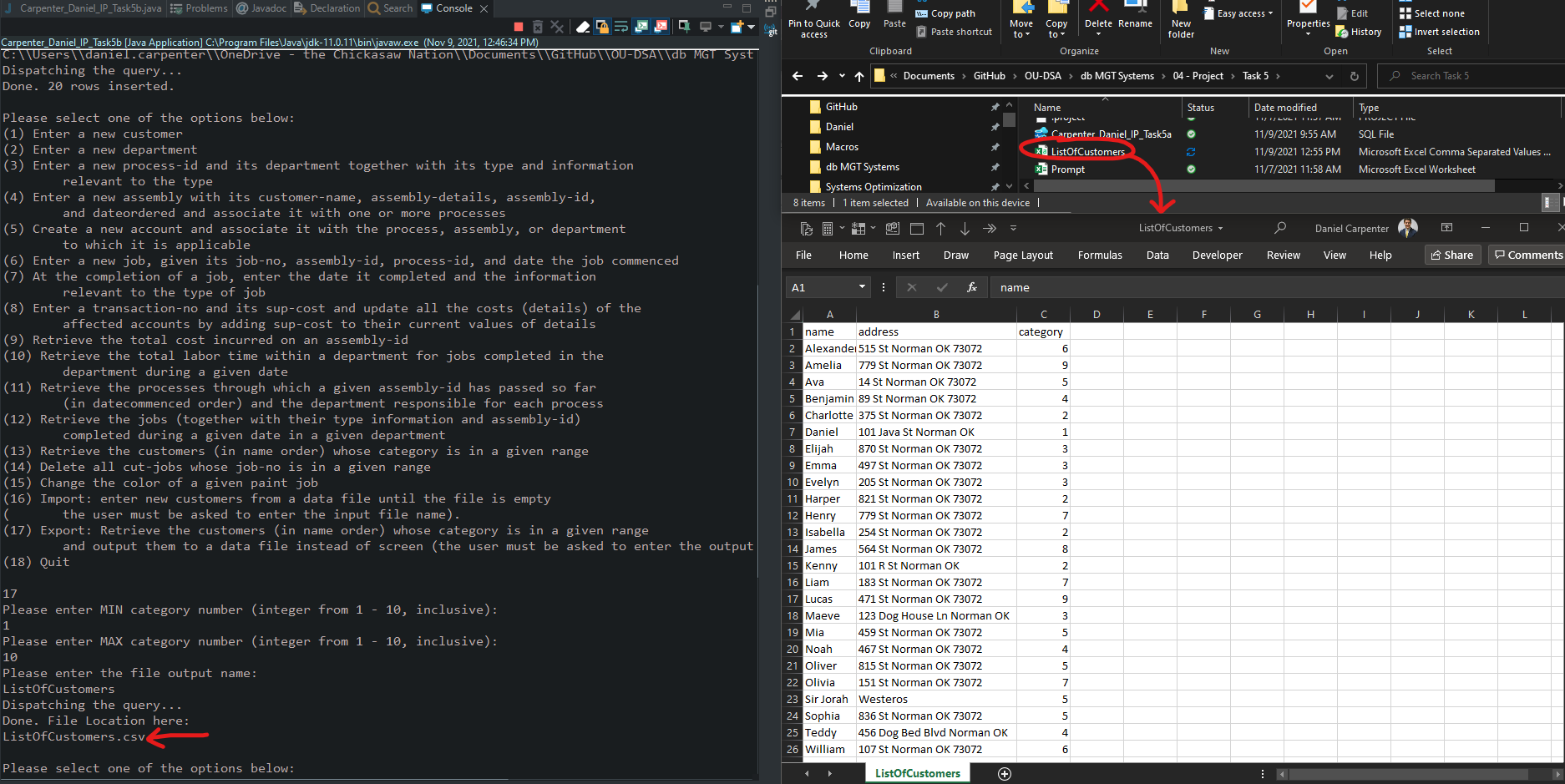
**Import Option (16) *(Continued)*:**

*Customer table after uploading file*

**

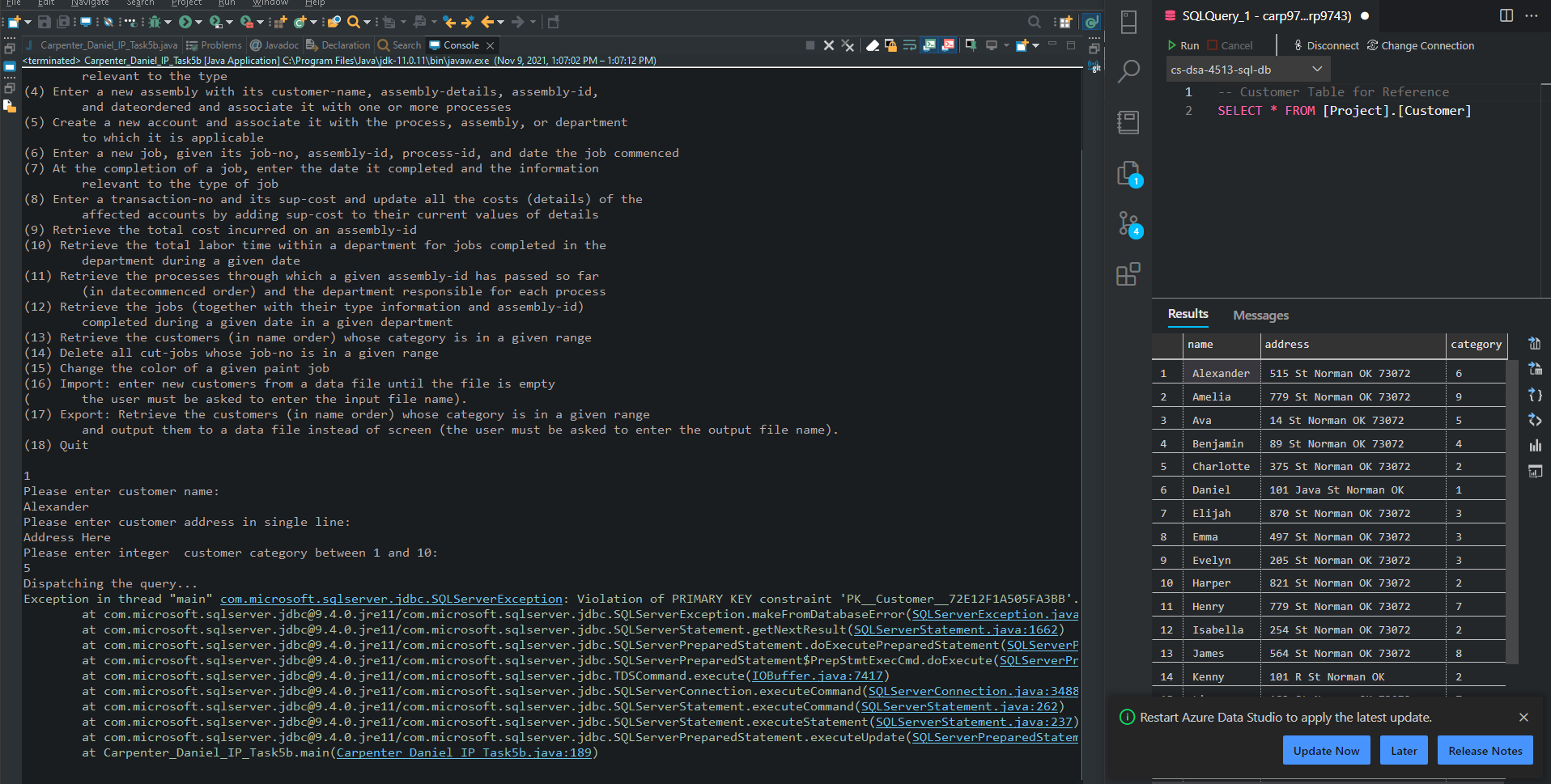
**6.16. Screenshots showing the testing of the import and export options *(Continued)***

**Export Option (17):**

****

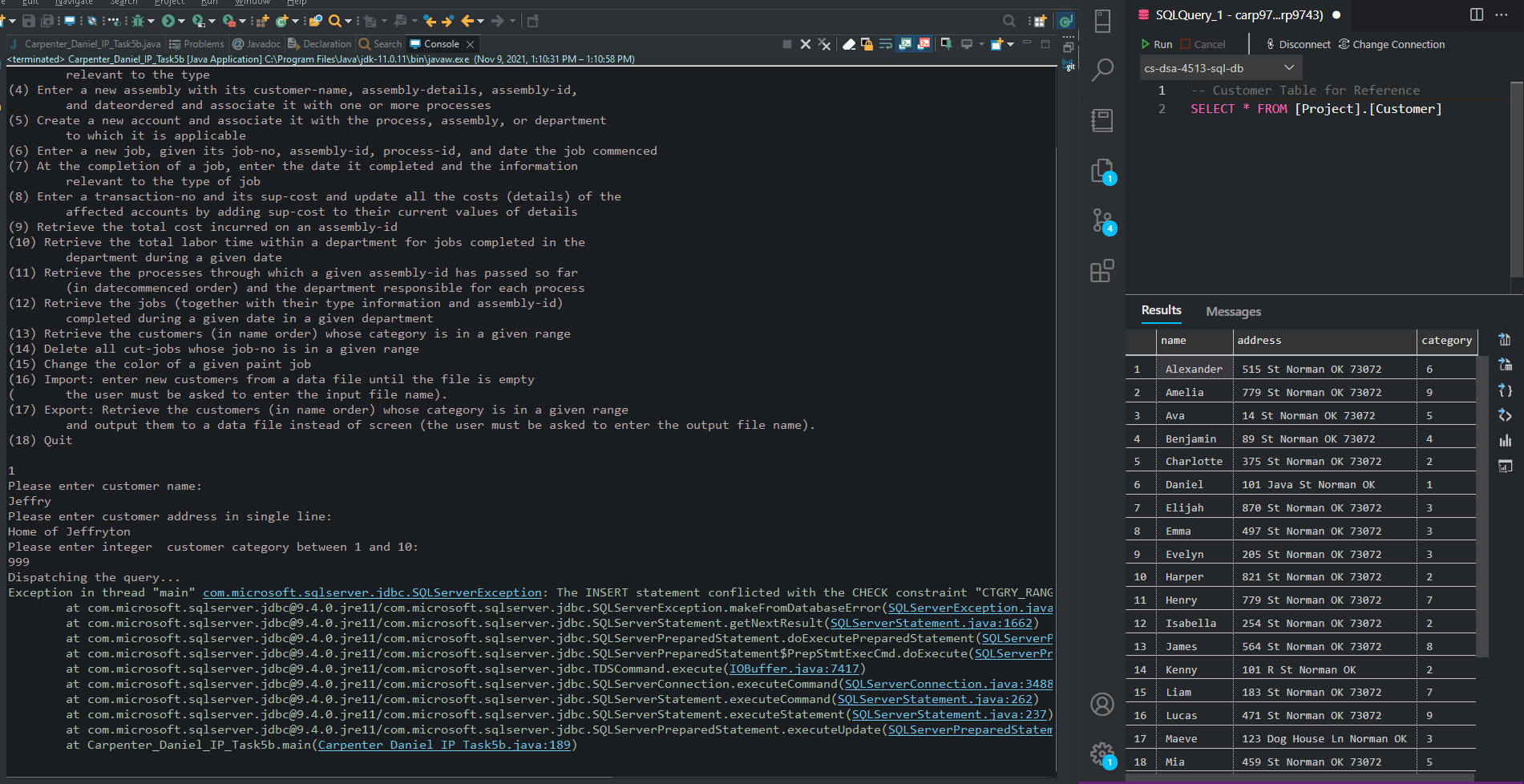
**6.17. Screenshots showing the Testing of Three Types of Errors**

**Error Example 1 – Violating a Primary Key constraint on the Table ‘Customer’:**

****

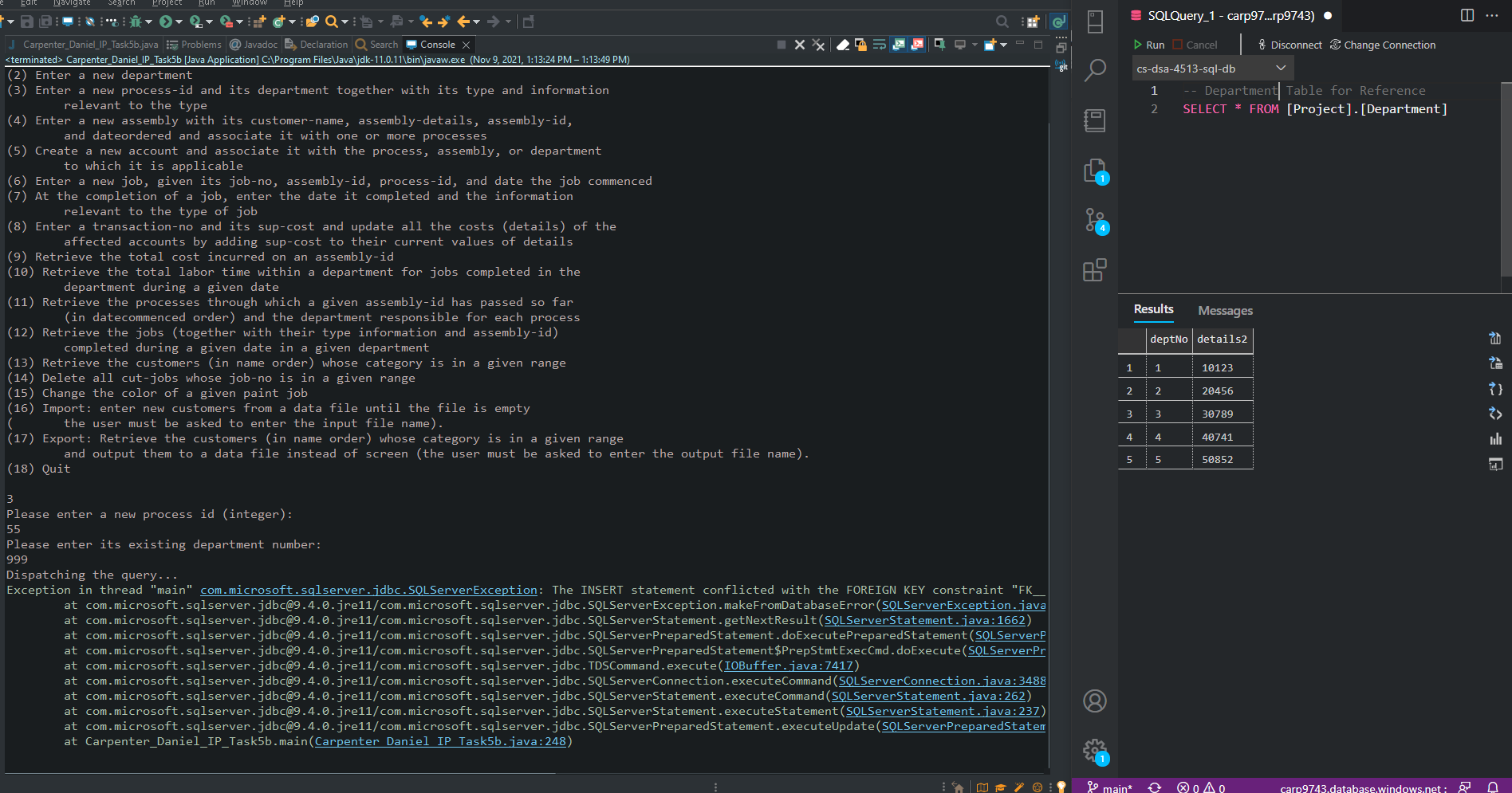
**6.17. Screenshots showing the Testing of Three Types of Errors *(Continued)***

**Error Example 2: Entering data outside of constraint range of table ‘Customer’ on attribute ‘category’. Expected values 1 through 10**

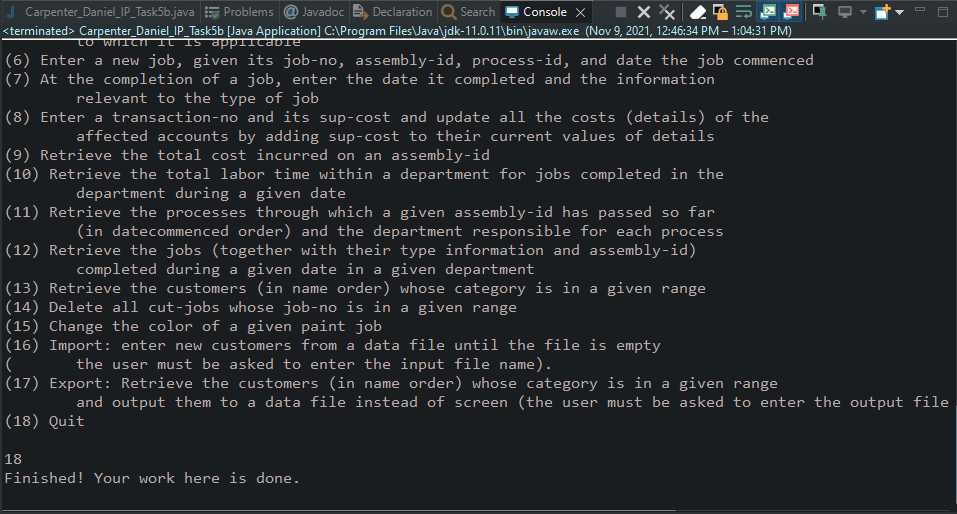
****

**6.17. Screenshots showing the Testing of Three Types of Errors *(Continued)***

**Error Example 3: Violating foreign key constraint by adding new Assembly with a non-existing ‘deptNo’ (which is the foreign key)**

****

**6.18. Screenshots showing the Testing of the Quit Option**

****

**7.1. Web Database Application Source Program**

**and Associated Screenshots of Successful Compilation**

***Carpenter\_Daniel\_IP\_Task7\_DataHandler.java***

|  |
| --- |
| package JobShopProject;  import java.sql.CallableStatement;  import java.sql.Connection;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  *//==========================================================================*  *//@class:  DSA 4513*  *//@asnmt:  Class Project*  *//@task:   7*  *//@author: Daniel Carpenter, ID: 113009743*  *//@description:*  *//  Program to implement job-shop accounting system query 1 and 13*  *//==========================================================================*  public class **Carpenter\_Daniel\_IP\_Task7\_DataHandler** {      private Connection conn;  *// Azure SQL connection credentials*      private String server   = "carp9743.database.windows.net";      private String database = "cs-dsa-4513-sql-db";      private String username = "carp9743";      private String password = "tacoBout$97315!";  *// Resulting connection string*      final private String url =              String.**format**("jdbc:sqlserver://%s:1433;database=%s;user=%s;password=%s;encrypt=true;trustServerCertificate=false;hostNameInCertificate=\*.database.windows.net;loginTimeout=30;",                      server, database, username, password);  *// Initialize and save the database connection*      private void **getDBConnection**() throws SQLException {          if (conn != null) {              return;          }          this.conn = DriverManager.**getConnection**(url);      }  *// Adds a customer to the Customer table and returns true if executed correctly*      public boolean **addCustomer**(String name, String address, int category) throws SQLException {  **getDBConnection**();    *// Prepare the query*          final String sqlQuery = "EXEC [Project].addCustomer @name = ?,  @address = ?, @category = ?;";    *// Replace the '?' in the above statement with the given attribute values*          final PreparedStatement statement = conn.**prepareStatement**(sqlQuery);              statement.**setString**(1,  name);              statement.**setString**(2,  address);              statement.**setInt**(3,     category);    *// Return true if successful*          return statement.**executeUpdate**() == 1;      }  *// Inserts a record into the movie\_night table with the given attribute values*      public ResultSet **getCustomersInRange**(int min, int max) throws SQLException {  **getDBConnection**(); *// Prepare the database connection*  *// Query*          String sqlQuery = "{CALL [Project].getCustomers(?, ?)}";    *// Prepare query call*          CallableStatement statement = conn.**prepareCall**(sqlQuery);    *// Set the assigned value(s) to the procedures input '?'*          statement.**setInt**("min", min);          statement.**setInt**("max", max);  *// Execute the query*          return statement.**executeQuery**();      }  } |

***Carpenter\_Daniel\_IP\_Task7\_getCustomersInRange.jsp***

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE *html* *PUBLIC* "-//W3C//DTD HTML 4.01 Transitional//EN"  "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta *http-equiv*="Content-Type" *content*="text/html; charset=UTF-8">  <title>Query Result</title>  </head>      <body>      <%@page import="JobShopProject.Carpenter\_Daniel\_IP\_Task7\_DataHandler"%>      <%@page import="java.sql.ResultSet"%>      <%@page import="java.sql.Array"%>      <%      // The handler is the one in charge of establishing the connection.      Carpenter\_Daniel\_IP\_Task7\_DataHandler handler = new Carpenter\_Daniel\_IP\_Task7\_DataHandler();      // Get the attribute values passed from the input form.      String min  = request.getParameter("min");      String max  = request.getParameter("max");       // Assume all categories if query fails       int minAsInt = 1;       int maxAsInt = 10;       ResultSet customers;        // detect input      if (min.equals("") || max.equals("")) {          response.sendRedirect("Carpenter\_Daniel\_IP\_Task7\_getCustomersInRange.jsp");      } else {        // Get the actual input        minAsInt = Integer.parseInt(min);        maxAsInt = Integer.parseInt(max);        }      // Now perform the query with the data from the form.      customers = handler.getCustomersInRange(minAsInt, maxAsInt);          %>  *<!-- The table for displaying all the customer records -->*      <table *cellspacing*="2" *cellpadding*="2" border="1">          <tr> *<!-- The table headers row -->*            <td align="center">              <h4>name</h4>            </td>          </tr>          <%             while(customers.next()) { // For each Customer record returned...                 // Extract the attribute values for every row returned                 final String name     = customers.getString("name");                   out.println("<tr>"); // Start printing out the new table row                 out.println( // Print each attribute value                      "<td align=\"center\">" + name + "</td>");                 out.println("</tr>");             }             %>        </table>        <a *href*="Carpenter\_Daniel\_IP\_Task7\_addCustomerForm.jsp">Add more customers.</a>      </body>  </html> |

***Carpenter\_Daniel\_IP\_Task7\_getCustomersInRangeForm.jsp***

|  |
| --- |
| <!DOCTYPE *html*>  <html>      <head>          <meta *charset*="UTF-8">          <title>Get customers whose category is in a given range </title>      </head>      <body>          <h2>Get Customers</h2>  *<!--*  *Form for collecting user input for the new Customer record.*  *Upon form submission, addCustomer.jsp file will be invoked.*  *-->*          <form *action*="Carpenter\_Daniel\_IP\_Task7\_getCustomersInRange.jsp">  *<!-- The form organized in an HTML table for better clarity. -->*              <table border=1>                  <tr>                      <th *colspan*="2">Retrieve the customers (in name order) whose category is in range 1 through 10 (integer):</th>                  </tr>                  <tr>                      <td>Min Value:</td>                      <td><div *style*="text-align: center;">                      <input *type*=text *name*=min>                      </div></td>                  </tr>                  <tr>                      <td>Max Value:</td>                      <td><div *style*="text-align: center;">                      <input *type*=text *name*=max>                      </div></td>                  </tr>                  <tr>                      <td><div *style*="text-align: center;">                      <input *type*=reset *value*=Clear>                      </div></td>                      <td><div *style*="text-align: center;">                      <input *type*=submit *value*=Submit>                      </div></td>                  </tr>              </table>          </form>      </body>  </html> |

***Carpenter\_Daniel\_IP\_Task7\_addCustomer.jsp***

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE *html* *PUBLIC* "-//W3C//DTD HTML 4.01 Transitional//EN"  "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta *http-equiv*="Content-Type" *content*="text/html; charset=UTF-8">  <title>Query Result</title>  </head>      <body>      <%@page import="JobShopProject.Carpenter\_Daniel\_IP\_Task7\_DataHandler"%>      <%@page import="java.sql.ResultSet"%>      <%@page import="java.sql.Array"%>      <%      // The handler is the one in charge of establishing the connection.      Carpenter\_Daniel\_IP\_Task7\_DataHandler handler = new Carpenter\_Daniel\_IP\_Task7\_DataHandler();      // Get the attribute values passed from the input form.      String name     = request.getParameter("name");      String address  = request.getParameter("address");      String category = request.getParameter("category");      /\*       \* If the user hasn't filled out all the fields. This is very simple checking.       \*/      if (name.equals("") || address.equals("") || category.equals("")) {          response.sendRedirect("Carpenter\_Daniel\_IP\_Task7\_addCustomerForm.jsp");      } else {          int categoryAsInt = Integer.parseInt(category);            // Now perform the query with the data from the form.          boolean success = handler.addCustomer(name, address, categoryAsInt);          if (!success) { // Something went wrong              %>                  <h2>There was a problem inserting the course</h2>              <%          } else { // Confirm success to the user              %>              <h2>The Customer:</h2>              <ul>                  <li>Customer Name: <%=name%></li>                  <li>Address: <%=address%></li>                  <li>Category: <%=categoryAsInt%></li>              </ul>              <h2>Was successfully inserted.</h2>                <a *href*="Carpenter\_Daniel\_IP\_Task7\_getCustomersInRangeForm.jsp">Retrieve list of customers.</a>              <%          }      }      %>      </body>  </html> |

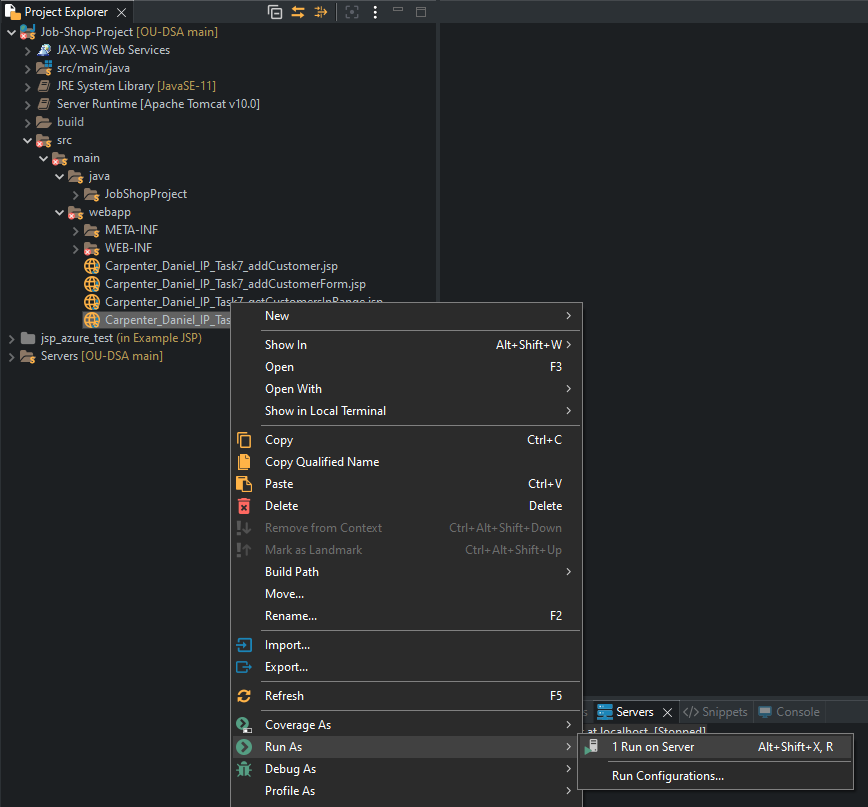
***Carpenter\_Daniel\_IP\_Task7\_addCustomerForm.jsp***

|  |
| --- |
| <!DOCTYPE *html*>  <html>      <head>          <meta *charset*="UTF-8">          <title>Add Customer</title>      </head>      <body>          <h2>Add Customer</h2>  *<!--*  *Form for collecting user input for the new Customer record.*  *Upon form submission, addCustomer.jsp file will be invoked.*  *-->*          <form *action*="Carpenter\_Daniel\_IP\_Task7\_addCustomer.jsp">  *<!-- The form organized in an HTML table for better clarity. -->*              <table border=1>                  <tr>                      <th *colspan*="2">Enter the Customer data:</th>                  </tr>                  <tr>                      <td>Customer Name:</td>                      <td><div *style*="text-align: center;">                      <input *type*=text *name*=name>                      </div></td>                  </tr>                  <tr>                      <td>Address:</td>                      <td><div *style*="text-align: center;">                      <input *type*=text *name*=address>                      </div></td>                  </tr>                  <tr>                      <td>Category:</td>                      <td><div *style*="text-align: center;">                      <input *type*=text *name*=category>                      </div></td>                  </tr>                  <tr>                      <td><div *style*="text-align: center;">                      <input *type*=reset *value*=Clear>                      </div></td>                      <td><div *style*="text-align: center;">                      <input *type*=submit *value*=Insert>                      </div></td>                  </tr>              </table>          </form>      </body>  </html> |

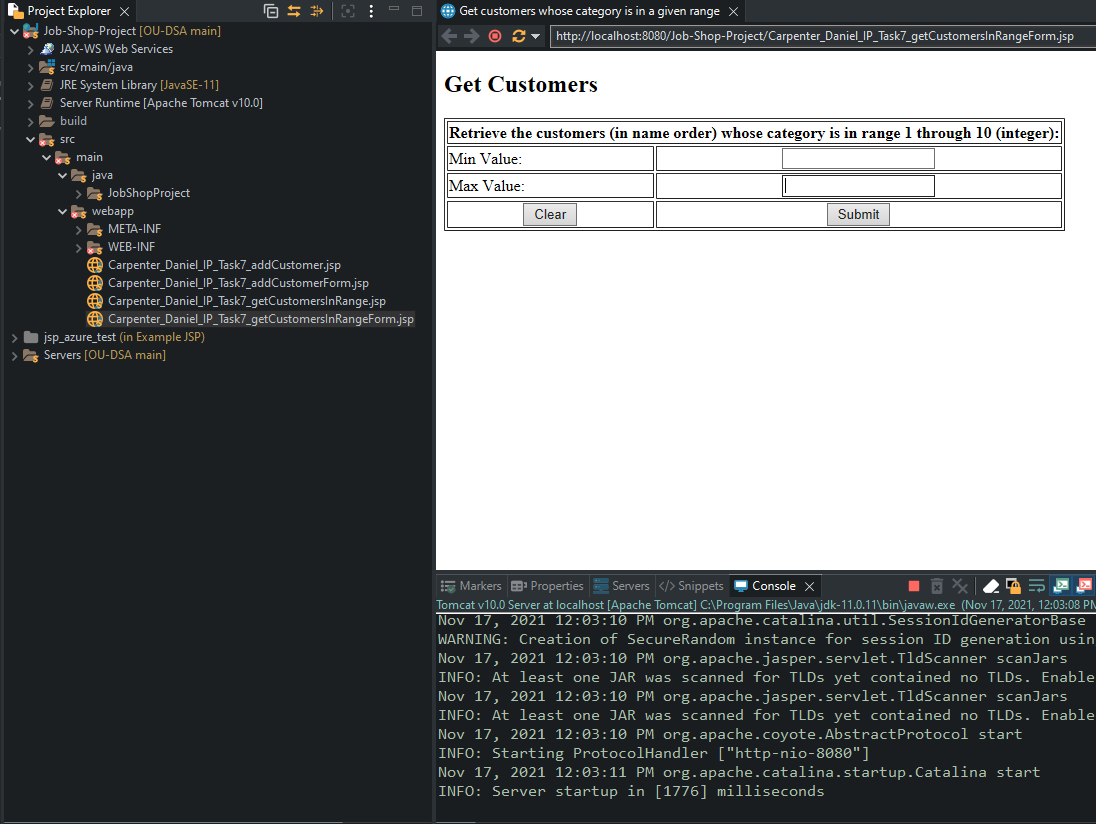
**7.1. Web Database Application Source Program**

**and Associated Screenshots of Successful Compilation *(Continued)***

**Prior to clicking Project**

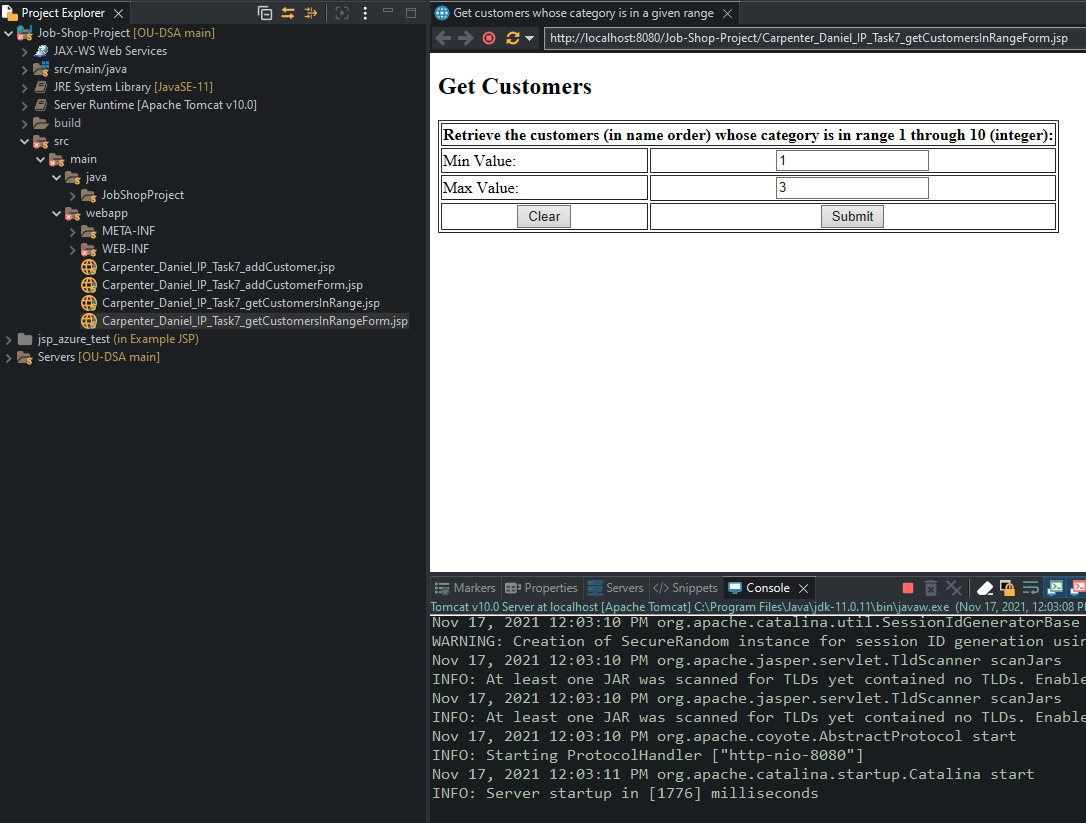
****

**Upon Clicking “Run on Server”**

****

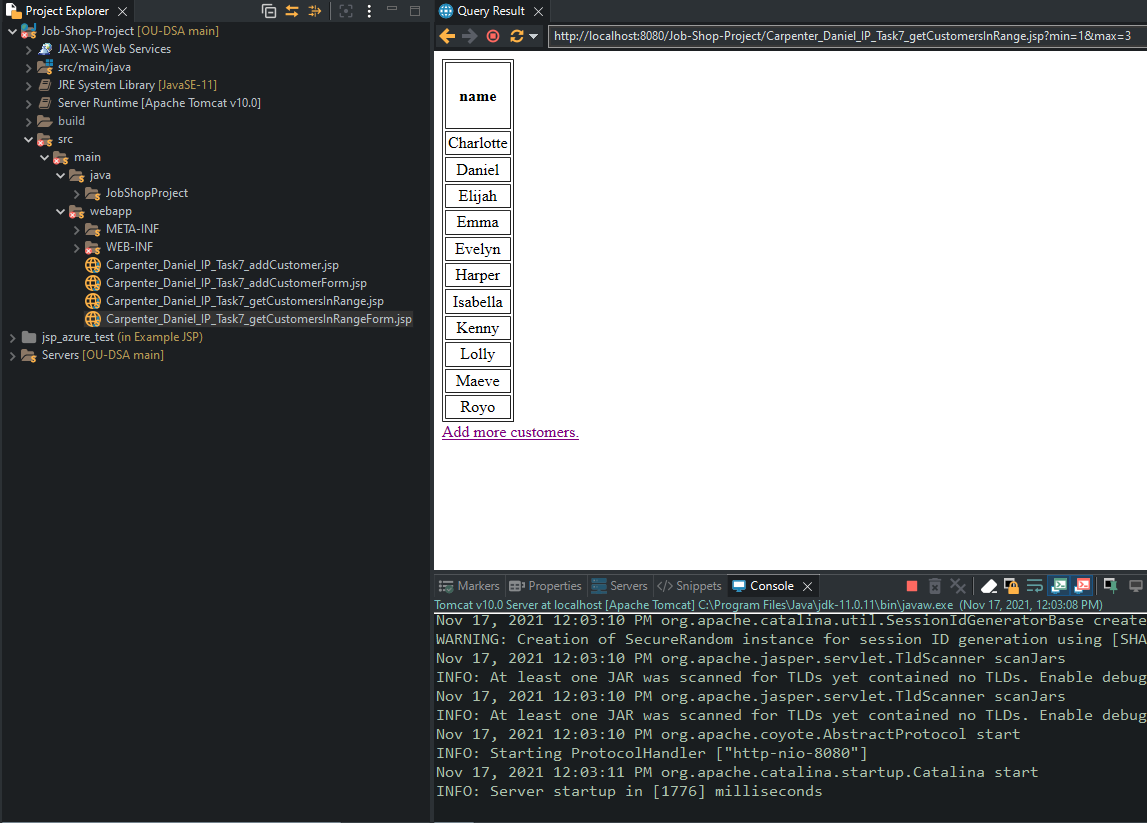
**7.2. Screenshots Showing the Testing of the Web Database Application**

**Screenshot 1: Application to get Customers,** *(with inputs range inputs from 1 to 5)*

****

**7.2. Screenshots Showing the Testing of the Web Database Application *(Continued)***

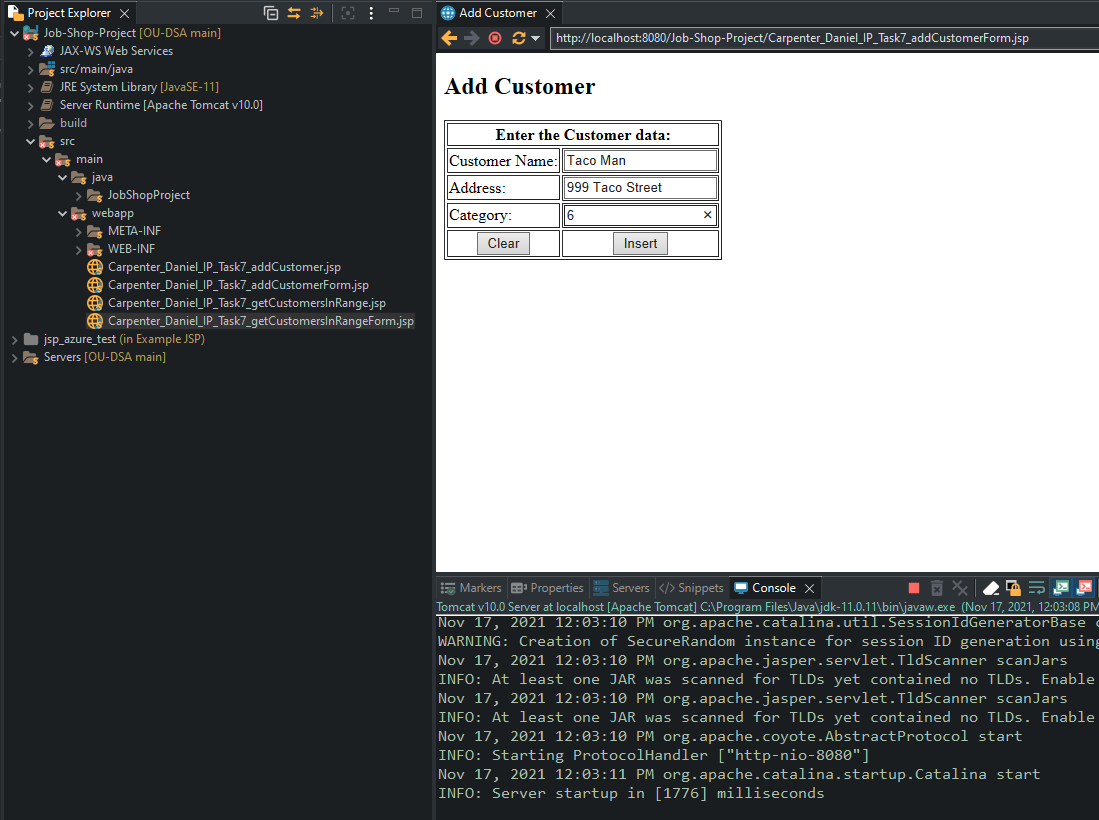
**Screenshot 2: Application after clicking “Submit” on the prior page**

****

**7.2. Screenshots Showing the Testing of the Web Database Application *(Continued)***

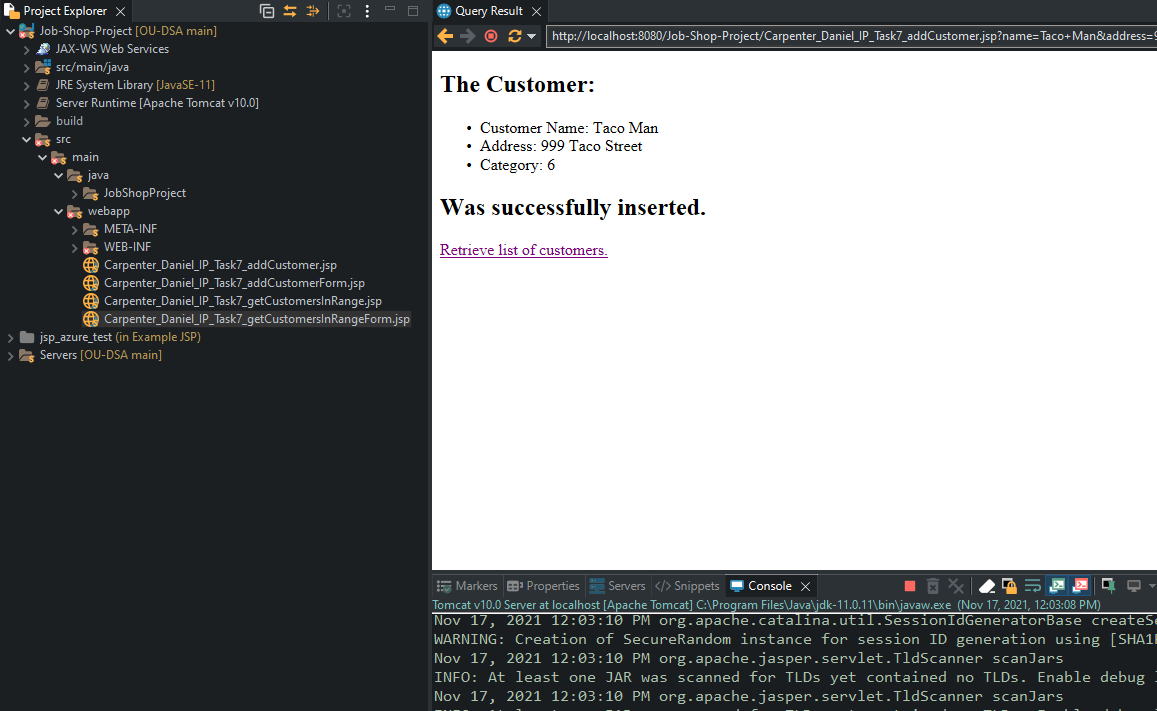
**Screenshot 3: Application after clicking the “Add more customers” button on prior page**

*(with relevant inputs to the Customer table)*

****

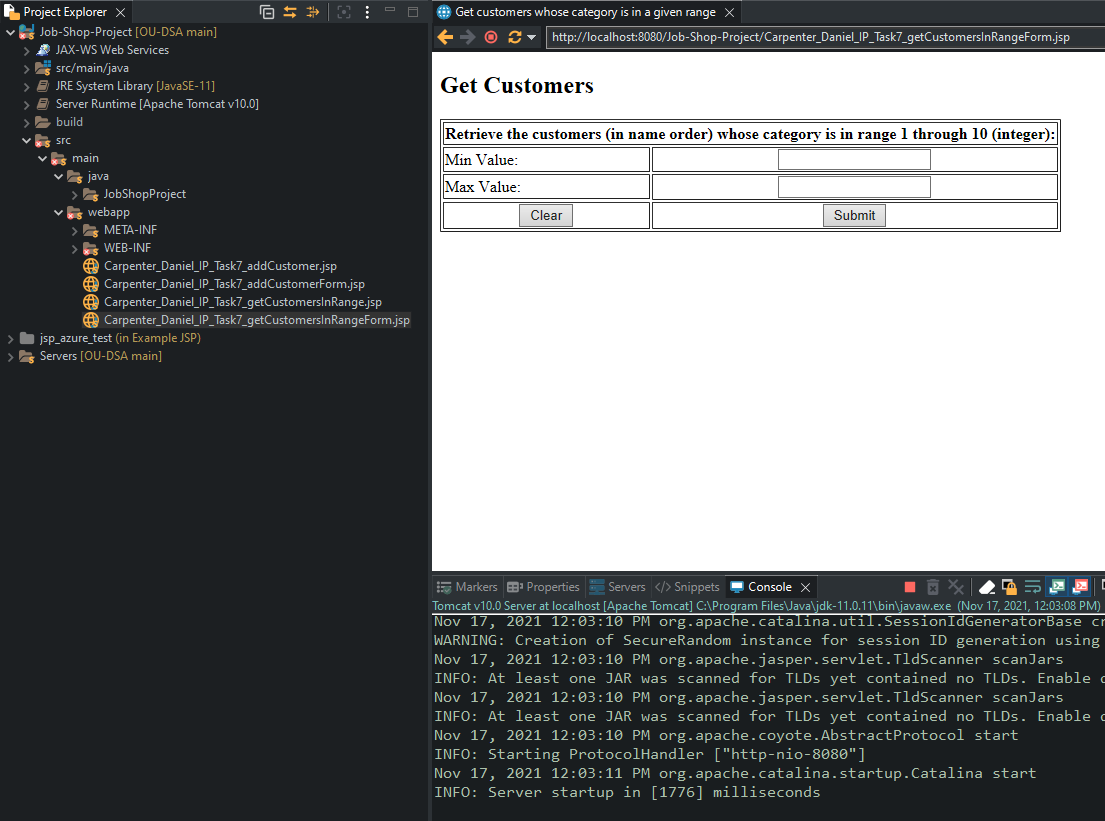
**7.2. Screenshots Showing the Testing of the Web Database Application *(Continued)***

**Screenshot 4: Application after clicking the “Insert” button on prior page**

****

**7.2. Screenshots Showing the Testing of the Web Database Application *(Continued)***

**Screenshot 5: Application after clicking the “Retrieve list of customers.” button on prior page**

****